

SPARE PARTS MASTER

CASIO®

SA02006410A * Printed in Japan

PB-120 OWNER'S MANUAL

CASIO

Personal computer
PB-120
OWNER'S MANUAL

CASIO

INTRODUCTION

This manual provides an explanation of the computer so that BASIC program beginners as well as users, who have a complete knowledge of BASIC and intend to fully utilize it, can easily understand and utilize the computer immediately.

Users who are new to BASIC programming should read this manual from Chapter 1 in order to master programming. Especially in Chapter 3, the explanation of program preparation and commands should be carefully read. A program flow explanation is provided in Chapter 3. See Chapter 5 "Command Reference" for the command formats and detailed explanation.

Users who have a knowledge of BASIC should utilize the computer while reading Chapter 5 "Command Reference" after mastering the basic operations explained in Chapters 1 and 2.

Users who intend to use DATA BANK FUNCTION immediately with programs can utilize the programs in Chapter 4 "Data Bank Function".

The contents of this manual may be subject to change without notice. Unlawful copying of all or any portion of this manual is strictly forbidden. Please be aware that the use of this manual for other than personal use without permission from CASIO is prohibited under the copyrighting law. CASIO Computer Co., Ltd. shall not be held responsible for any damages or losses resulting from the use of this manual. Furthermore note that CASIO assumes no responsibility for any loss or claims by third parties which may arise through use of this unit.

CONTENTS

BEEP	165
DEFM	166
MODE	167
SET	168
LEN	169
MID\$	170
VAL	171
STR\$	172
SIN, COS, TAN	173
ASN, ACS, ATN	174
LOG, LN	174
EXP	175
SQR	175
ABS	176
SGN	176
INT	176
FRAC	177
RND	177
RAN#	178
DEG	178
DMS\$	179
DATA BANK COMMANDS	180
NEW#	180
LIST#	180
SAVE#	181
LOAD#	181

CONTENTS

READ#	182
RESTORE#	183
WRITE#	185

CHAPTER 6 REFERENCE MATERIAL

6-1 ERROR MESSAGE TABLE	188
6-2 CHARACTER CODE TABLE	189
COMMAND/FUNCTION INDEX	190
SPECIFICATIONS	191

CONTENTS

CONTENTS

CHAPTER 1 GENERAL GUIDE

1-1	NOMENCLATURE AND OPERATION	10
1-2	ABOUT THE POWER SUPPLY	16
1-3	MEMORY EXPANSION	18
1-4	BEFORE CALCULATING	19

CHAPTER 2 LET'S OPERATE

2-1	LET'S OPERATE THE COMPUTER	22
2-2	CONVENIENT DATA BANK FUNCTION	26
2-3	SIMPLE CALCULATION, AT THE BEGINNING	27
2-4	FUNCTION CALCULATION — A HIGHLIGHT OF THIS COMPUTER	29

CHAPTER 3 "BASIC" PROGRAMMING

3-1	WHAT IS A PROGRAM?	36
3-2	PROGRAM PREPARATION	39
3-3	PROGRAM DEVELOPMENT	58
3-4	CONVENIENT OPTIONAL EQUIPMENTS	90
3-5	USING A PB-100 PROGRAM	97

CHAPTER 4 DATA BANK FUNCTION

4-1	HOW TO USE THE DATA BANK FUNCTION	102
4-2	HOW TO USE THE DATA BANK FUNCTION IN A PROGRAM	119
4-3	EXAMPLES OF DATA BANK PROGRAM	124

CHAPTER 5 COMMAND REFERENCE

NEW [ALL]	143
-----------------	-----

RUN	144
LIST	145
PASS	146
SAVE [ALL]	147
LOAD [ALL]	148
VERIFY	149
CLEAR	150
END	150
STOP	150
LET	151
REM	151
INPUT	152
KEY\$	153
PRINT	154
CSR	155
GOTO	156
ON — GOTO	157
IF — THEN	158
FOR — NEXT	159
GOSUB	160
RETURN	160
ON — GOSUB	161
DATA	162
READ	163
RESTORE	164
PUT	164
GET	164

PRIOR TO OPERATION

This computer was delivered to you through CASIO's strict testing process, high level electronics technology, and strict quality control. To ensure a long life for your computer, please observe the following precautions.

■ Precautions

- Since this computer consists of precision electronic parts, do not disassemble it. Also do not apply an impact to it by throwing or dropping it, or do not expose it to rapid temperature changes. In addition, do not store it in a place with high temperatures or high humidity, or in a dusty place. When the computer is utilized in low temperatures, sometimes the display response is slow or does not operate. When normal temperature conditions are restored, however, the computer operation will become normal.
- Special care should be taken not to damage the computer by bending. For example, do not carry it in your hip pocket.
- Please do not connect units other than the FA-3, FA-5, FP-12S and SB-42 to the connector portion.
- Since "—" is displayed during calculation in which key operation is invalid except for certain keys, always confirm the display before pressing a key.
- Although the display sometimes becomes faint while buzzer is sounding, it is not a malfunction. However, if the display becomes very faint, replace the batteries with new ones as soon as possible.
- Every two years, replace the batteries with new ones even if the computer is not utilized. Do not leave exhausted batteries inside it because trouble may occur due to battery leakage.
- Always keep the cap for the connector portion when only the computer is used.
- If strong static electricity is applied to the computer, sometimes the memory content is changed, or key operation cannot be performed. If this occurs, remove the batteries, then replace them again.

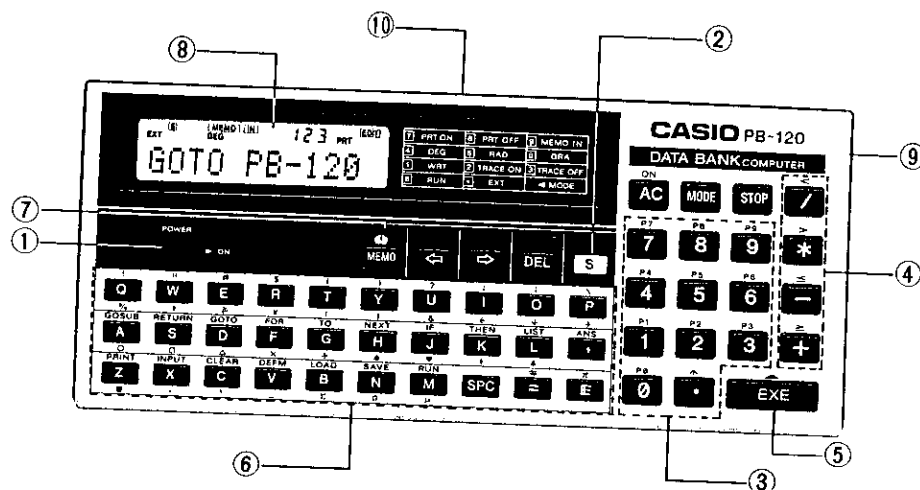
- Always connect optional equipments after turning the computer power off.
- To clean the computer, do not use volatile liquids such as benzine or thinner, but wipe it with a soft dry cloth, or a cloth dampened with a neutral detergent solution.
- Do not turn the power off during program execution or operation.
- Since the computer is made up of precision electronic parts, avoid giving a strong shock while a program is being executed; otherwise the program execution may be stopped or the memory contents may be changed.
- When a malfunction occurs, contact the store where the computer was purchased or a nearby dealer.
- Before seeking service, please read this manual again, check the power supply, check the program for logic errors, etc.

CHAPTER 1

GENERAL GUIDE

Users who have not used a computer as well as those who are accustomed to a computer should read this chapter.

1-1. NOMENCLATURE AND OPERATION

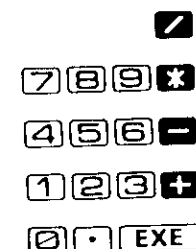


- ① Power switch
- ② Shift key
- ③ Numeral and decimal point keys
- ④ Calculation keys
- ⑤ Execution key
- ⑥ Alphabetical keys
- ⑦ Memorandum key
- ⑧ Display window
- ⑨ Display contrast control
- ⑩ Connector portion

1-1. NOMENCLATURE AND OPERATION

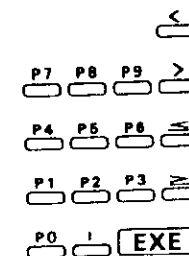
Since many additional keys are provided compared to an ordinary calculator, the key functions might be unclear. Therefore each key and operation are explained.

- **Power switch**
When this switch is moved to the right, the power is turned on, and when it is moved to the left, the power is turned off.
- **Shift key (Red $\boxed{S}$ key)**
If this key is pressed, the shift mode is selected (" $\boxed{S}$ " is displayed) and the command or symbol printed above each key can be displayed. When it is pressed again, the shift mode is released and " $\boxed{S}$ " disappears. (To distinguish this key from the alphabetical $\boxed{S}$ key, it will be written as $\boxed{SHIFT}$ from now on in this manual.)
- **Numeral keys, decimal point key, calculation keys, and execution key**
Examine this key array carefully. It is the same as that of an ordinary calculator, isn't it? This part is used when the four arithmetic calculations (addition, subtraction, multiplication, division) are performed. However, the following differences exist. The $\times$ (multiplication) and $\div$ (division) keys are different and there is no $=$ key while there is an $\boxed{EXE}$ (execution) key. This occurs because a computer uses an $*$ (asterisk) for $\times$ and $/$ (slash) for $\div$, while the answer is obtained by the $\boxed{EXE}$ key instead of the $\boxed{=}$ key.



For example, an operation is performed by an ordinary calculator as $12 \times 4 \div 3 + 7 - 5 =$ while this computer uses $12 * 4 / 3 + 7 - 5 \boxed{EXE}$.

This computer can be used as an ordinary calculator as shown above. When followed by the $\boxed{SHIFT}$ key, one of the numeral keys ($\boxed{0}$ to $\boxed{9}$) can be used to specify a program area from P0 to P9 while the $\boxed{\square}$ key is used for power calculation ($x^y \rightarrow x \uparrow y$) and the $\boxed{\geq}$, $\boxed{\leq}$, $\boxed{>}$, $\boxed{<}$ keys are used to enter relational operators ($\geq, \leq, >, <$).



- **Alphabetical keys, space key**

Q W E R T Y U I O P
A S D F G H J K L ,
Z X C V B N M SPC = E

Using these keys, commands are entered, or programs are written. Each of the 26 alphabetical keys from **A** to **Z** functions as a memory (for storage locations).

Also, the **[A] — [Z]** keys have another function. When they are pressed after the **[SHIFT]** key, a symbol or BASIC command is displayed. Press the **[SPC]** key when a space is required.

Example) **SHIFT** **A** → GOSUB、 **SHIFT** **U** → ?

1 2 3 4 5 6 7 8 9 0
GOSUB RETURN GOTO FOR TO NEXT IF THEN LIST ANS
PRINT INPUT CLEAR DEFN LOAD SAVE RUN SPC DEL π

In addition, the alphabetical keys have another use in the extension mode (When  is pressed after the **MODE** key, "EXT" is displayed). When they are directly pressed, small alphabetical characters are displayed, and when they are pressed after the  key, special symbols are displayed.

Extension mode functions:

q w e r t y u i o p
a s d f g h j k l .
z x c v b n m SPC = E

Functions provided when a key is pressed after the  key in the extension mode:

When a key is

	.	@	v	(	)	&	-	i	=
				x	+	◆	♥	♦	ANS
O		Δ							
	.	o	-	Σ	Π	μ	SPC	≠	π

To release the extension mode, press **MODE**  again.

- Equal key (**E**)

Equal key (=)
This key is not used to provide an answer for calculation, but is used for an assignment statement (see page 40) and for a condition in an IF statement (see page 62).

Also, when this key is pressed after the  key, a $\neq$ (not equal) symbol is displayed.

- Exponent/Pi key ($\frac{\pi}{x}$)

Exponent/Pi key ($\frac{x}{y}$)
When this key is directly pressed, it is used to provide an exponent. For example, operate $1 \div 2 \div 3 \div 4$ for 1.23×10^4 . When an exponent is a negative number, press the $\pm$ key after this key. For example, operate $7 \div 4 \div 1 \div \pm$ for 7.41×10^{-9} .

When this key is pressed after the  key, Pi (the ratio of the circumference of a circle to its diameter) is displayed.

- Answer key ()

Answer key ()
When this key is pressed after the  key, the result of manual or program calculation executed immediately before is displayed.

- Mode key ()

Mode key ()
This key is used together with  and  to  when the computer status or angle unit is specified.

[MODE] []..... "EXT" is displayed to indicate extension mode in which small alphabetical characters and special symbols can be used. To release the extension mode, press these keys again.

MODE  "RUN" is displayed for the performance of manual and program calculations.

- MODE** **(1)**..... "WRT" is displayed for the performance of program write-in, checking, and editing.
- MODE** **(2)**..... "TR" is displayed for the performance of execution trace. (see page 57 for details.)
- MODE** **(3)**..... When "TR" is displayed, execution trace mode is released and "TR" disappears.
- MODE** **(4)**..... "DEG" is displayed to indicate that degree is specified as the angle unit.
- MODE** **(5)**..... "RAD" is displayed to indicate that radian is specified as the angle unit.
- MODE** **(6)**..... "GRA" is displayed to indicate that grade is specified as the angle unit.
- MODE** **(7)**..... "PRT" is displayed for the performance of printing when a printer is connected.
- MODE** **(8)**..... When "PRT" is displayed, print mode is released and "PRT" disappears.
- MODE** **(9)**..... "**MEMO** **IN**" is displayed to indicate input mode for the Data Bank function (see page 102). To release this mode, press **MODE** **(0)**.

- Memorandum key ()

Pressed to use the Data Bank function. Also pressed for sequential recall or for recall after pressing a specified character in the RUN mode (press **MODE** **□**) or in the input mode (press **MODE** **9**).

- **Cursor keys** ()

These keys are used to move the cursor (blinking “_” in the display window) to the left or right as a convenience when correcting a displayed character. When they are pressed once, the cursor is moved one character, and when they are continuously pressed, the cursor moves continuously within the range of written characters.

- All clear key (**AC**)

This key erases any display. Also, it is pressed when an error occurs, or when the display blanks out by auto power off (see page 17). When a program is being executed, program execution is suspended by pressing this key.

- Delete/Insert key ()

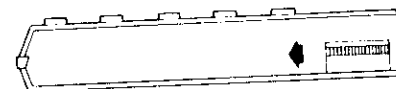
Delete/Insert key ()
This key is used to delete a character where the blinking cursor is positioned. After deletion, the character to the right of the cursor moves to the left. When it is pressed after the  key, the character where the blinking cursor is positioned is moved to the right to provide a space.

- Stop key ()

- **Stop key (STOP)**
When this key is pressed during program execution, it is temporarily stopped. To resume it, press the **EXE** key.

- **Display contrast control**

Display contrast control
When the display is dark or faint, depending on the battery condition or display view angle, adjust it by moving the control located on the right side of the computer.



The display becomes darker when the control is turned in the direction of the arrow, and becomes lighter when turned in the opposite direction. If the display is still faint when this control is placed in the darkest position, the batteries are weak and should be replaced with new ones.

- Connector portion

Connector portion
This connector is used when an optional equipment is connected. When printing is required, the FP-12S or SB-42 (for FP-40) is connected to it, and when program storing on a tape required, the FA-3 or FA-5 is connected to it.



Do not connect any equipment other than the FA-5, FA-3, FP-12S and SB-42 (for FP-40) to this connector portion. When these optional equipments are not connected, always place the attached connector cap on it.

1-2. ABOUT THE POWER SUPPLY

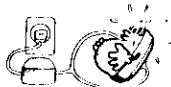
Power is supplied by two CR2032 lithium batteries for normal operation, as well as one CR1220 lithium battery for memory backup. Programs and data will be lost if all of the batteries are removed at the same time.

- * Whenever all of the batteries have been removed from the unit for some reason, press the "All Reset" button after loading batteries and switching power ON.

Precautions

Incorrectly using batteries can cause them to burst or leak, possibly damaging the interior of the unit. Note the following precautions:

- * Be sure that the positive (+) and negative (-) poles of each battery are facing in the proper direction.
- * Never mix batteries of different types.
- * Never mix old batteries and new ones.
- * Never leave dead batteries in the battery compartment.
- * Remove batteries if you do not plan to use the unit for long periods.
- * Replace batteries at least once every 2 years, no matter how much the unit is used during that period.
- * Never try to recharge the batteries supplied with the unit.
- * Do not expose batteries to direct heat, let them become shorted, or try to take them apart.



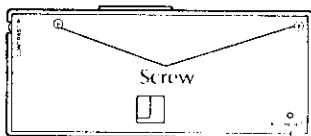
Keep batteries out of the reach of small children. If swallowed, consult with a physician immediately.

■ Main power supply batteries

The standard life of the main batteries is approximately 130 hours (for continuous program execution), but actual life may be somewhat shorter, depending upon how much the buzzer is used. If the display is dim, even after you adjust the contrast to a lighter setting (see page 15), replace batteries as soon as possible.

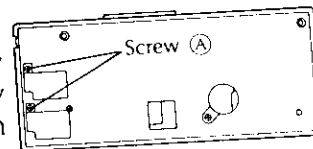
To replace the main power supply batteries

- (1) Switch the power of the unit OFF. Remove the two screws that hold the back cover of the unit in place, and then remove the back cover.
- (2) Remove the screws (A) from the main power supply battery holders, and then remove the battery holders.

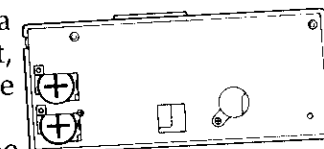


All Reset button

- (3) Remove the old main power supply batteries from the unit. This can be accomplished by positioning the unit so that the battery compartment is facing downwards, and then lightly tapping the unit.



- (4) Wipe the surface of two new batteries with a soft, dry cloth, and place them into the unit, making sure that their positive (+) sides are facing upwards.



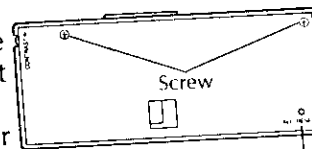
- (5) Fasten the battery holders in place using the screws.

- (6) Replace the back cover.

- Never switch the power of the unit ON while replacing batteries. Doing so will damage memory contents.
- Be sure to replace batteries at least once every two years, regardless of how much the unit is used.

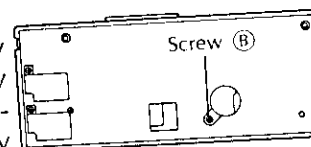
To replace the memory backup battery

- (1) Switch the power of the unit OFF. Remove the two screws that hold the back cover of the unit in place, and then remove the back cover.

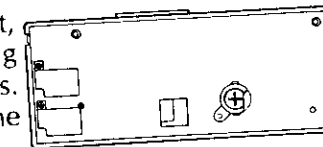


All Reset button

- (2) Remove the screw (B) from the backup power supply battery holder, and then remove the battery holder.



- (3) Remove the old backup power supply battery from the unit. This can be accomplished by positioning the unit so that the battery compartment is facing downwards, and then lightly tapping the unit.



- (4) Wipe the surface of a new battery with a soft, dry cloth, and place it into the unit, making sure that its positive (+) side is facing upwards.
- (5) Fasten the battery holder in place using the screw.

- (6) Replace the back cover.

- Memory contents are protected by the main power supply batteries when the backup power supply battery is removed.
- Be sure to replace batteries at least once every two years, regardless of how much the unit is used.

- The battery that comes with the unit was loaded at the factory, and does not provide the battery life noted in the specifications. Be sure to replace this battery as soon as the display of the unit appears dim.

■ Auto power off

The auto power off function prevents wasted power consumption when you forget to turn off the power switch. The power automatically turns off about 6 minutes after the last key operation (excluding program calculation). In this case, the power is turned on again by turning the power switch off and then on, or by pressing the **ON** key.

* Although the memory contents are not erased when the power is turned off, the angle and mode specifications ("RAD", "WRT", "TR", "PRT", etc.) are released.

1-3. MEMORY EXPANSION

There are normally 26 memories (variables). The number of steps at this time is 7648. The maximum number of standard memories is 982.

Number of Memories	Number of Program Steps
	Standard
26	7648
27	7640
28	7632
⋮	⋮
46	7488
⋮	⋮
982	0

Memory expansion is performed in units of 1 using a DEFM command.

Example)

Expand by 30 and make 56.

Operation)

Select the RUN mode (press **MODE** **(3)**) or the WRT mode (press **MODE** **(1)**).

DEFM 30 **EXE**

***VAR:56

*DEFM can be input by pressing **[D][E][F][M]** or by pressing **[SHIFT][V]**.

A DEFM command is also used to confirm the number of memories which are currently designated.

Example)

A total of 56 memories are designated.

DEFM

EXE

***VAR:56

- When a large number of program steps are already in use, in order to protect the existing program, if a designation is attempted which would cause an insufficient number of steps, an error will occur. (ERR 1 insufficient number of steps)
- The exclusive character variable (\$) is not counted when designating since it is a special memory.

1-4. BEFORE CALCULATING

■ Calculation priority sequence

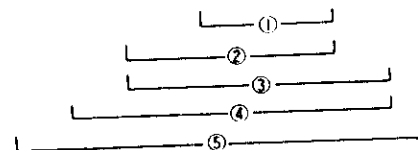
Calculations have "priority sequence" rules in which multiplication and division are performed prior to addition and subtraction. This computer is provided with a function that automatically distinguishes the priority sequence. This function is so convenient that a correct answer can be obtained by entering a calculation expression as it is.

The calculation priority sequence is determined as follows.

- ① Functions (SIN, COS, etc.)
- ② Power (↑)
- ③ $\times$ (*), $\div$ (/)
- ④ +, -

Although calculation is performed according to this priority sequence, if two operations have the same priority, the left one has priority. If parentheses are used, operations inside parentheses have priority.

Example) $2 + 3 * \sin(17 + 13) \uparrow 2 = 2.75$ (DEG MODE)



■ Input/output number of digits and operation number of digits

The number of input digits are 12 digits for a mantissa and 2 digits for an exponent. Internal operations are also performed using 12 digits for a mantissa and 2 digits for an exponent.

Although the number of output digits is usually 10 digits for an mantissa, it differs depending on a displayed result of a manual calculation and that of a program calculation. In the manual calculation, the result is displayed up to 12 digits including mantissa, exponent and minus sign. While in the program calculation, 10 digit mantissa and 2 digit exponent are displayed. However, if 12 digits are exceeded, 12 digits from the beginning are displayed first, then the rest is displayed sequentially by shifting the display to the left.

Example)

Manual calculation

1 □ 2345678912 **EXI**
 12345678912 **×** 100 **EXI**
 12345678912 **÷** -100 **EXI**

1.234567891
1.2345678 E 12
-1.234567 E 12

Program calculation

For PRINT 12345678912 **÷** -100

-	-1.234567891	E 12
-1	1.234567891 E	12
-1.	.234567891 E 1	2
-1.	234567891 E 12	

Disappears from
the display.

Has not been
displayed.

Is automa-
tically shifted.

CHAPTER 2

LET'S OPERATE

You should operate this computer to become accustomed to it because it won't break even if it is wrongly operated. First, try a simple operation in accordance with the proverb "Practice makes perfect."

2-1. LET'S OPERATE THE COMPUTER

Learn how to operate the computer by actually using it. To start, hold the computer and turn the power on by sliding the power switch to the right. Then the following is displayed.

RUN DEG
READY P0

Erase this display first by pressing the **AC** key. "READY P0" disappeared, didn't it? When this occurs, " _ " blinks on the extreme left of the display. This is called a "cursor" where a character can be written.

RUN DEG
_

When the cursor blinks, this is called an "input wait state" in which the computer waits for the input of calculation or an instruction. While the cursor usually blinks as " _ ", it also blinks as " █ " while characters are continuously written. Up to 62 characters can be written on one line. When 56 or more characters are written, the " █ " sign appears as a warning signal. "RUN" and "DEG" on the display indicate the present status. "RUN" indicates the RUN mode in which manual calculations or program executions can be performed. "DEG" indicates that the angle unit is degree.

The angle unit also includes the Radian mode ("RAD" is on) which is specified by pressing **MODE** **5**, and the Grade mode ("GRA" is on) which is specified by pressing **MODE** **6** in addition to the above. The angle units are required when trigonometric functions are used. Whenever the power switch is turned on, "DEG" is displayed.

The status display also includes the program write-in mode ("WRT" is on) specified by pressing **MODE** **1**, the trace mode ("TR" is on, see page 57) specified by pressing **MODE** **2**, the print mode ("PRT" is on, see page 95) by pressing **MODE** **7**, the data input mode for DATA BANK function ("MEMO IN" is on, see page 102) by pressing **MODE** **9**, and the extension mode ("EXT" is on) by pressing **MODE** **4**.

You will learn about these display modes as you continue to operate this computer.

Let's actually operate the computer to understand the display. If a message stays displayed after using the corresponding mode, turn the power switch off and then turn it on again.

Start with a simple calculation.

Example) $123+456=579$

_

Press **AC**.

Press appropriate keys to enter the numerical expression.

123+456

123+456_

After this, an answer is obtained by pressing **EXE** instead of **=**.

EXE

579

Next, try another calculation.

Example) $45 \times 6 + 89 = 359$

Consider that 45 was pressed as 46 by mistake.

46*6+89

46*6+89_

Now you notice that 46 was pressed by mistake. Place the cursor at the location where the wrong key was pressed by using a cursor key (**←**) calmly.

←←←←←

4*6+89
↑
The cursor and 6 turn on and off.

Press the correct key, **5**.

5

45*6+89

Since the calculation expression is now corrected by performing the above procedure, press the **EXE** key to obtain the answer.

EXE

359

When a mistake was found in the middle as mentioned above, it can be easily corrected by using the cursor keys. However, if the **EXE** key has been pressed, reenter the calculation expression from the beginning.

Now, let's enter characters by using the alphabet keys. The array of the alphabet keys is the same as that of a QWERTY typewriter. Most personal computers now have QWERTY type array keyboards; remember the location of the characters even though it may not be easy for a beginner to do this.

Enter characters first.

Example) The characters used are "ABCXYZ".

Enter ABC.

A B C

ABC_

Enter XYZ next.

X Y Z

ABCXYZ_

Now put a space between ABC and XYZ.
Place the cursor on X.

← →

ABCXYZ

Make one character space.

SHIFT INS
(DEL)

ABC_XYZ

When a space is to be inserted between characters, place the cursor at the location where it is to be inserted, then press **SHIFT INS** (DEL).
When additional spaces are to be inserted, repeat this procedure.

This computer is provided with some special characters which are convenient for games or as scientific symbols in addition to alphanumeric characters. (See page 12 for the special characters.)

Let's practice displaying some of these characters.

Example) Display ♠♥♦♣ marks.

Specify the extension mode.

AC MODE

Displayed
_

For these marks, press alphabet keys after pressing the **SHIFT** keys.

SHIFT ♠ SHIFT ♥ SHIFT ♦ SHIFT ♣

♠♥♦♣_

Example) Display Σ, Ω, μ symbols.

Since the extension mode is used, just perform the following operations.

SHIFT Σ SHIFT Ω SHIFT μ

♠♥♦ΣΩμ_

Since these marks and symbols are provided, please try to use them. Also, to return to the mode in which upper case letters are displayed, press **MODE** again to erase "EXT". During the continuous use of this computer, sometimes "ERR2" is displayed and it does not operate even if a key is pressed. This is not computer trouble, but is a message called an "error message". When this occurs, press the **AC** key, then the display clears and the computer operates again. See pages 52 and 188 for details.

2-2. CONVENIENT DATA BANK FUNCTION

This computer is provided with the DATA BANK function which allows data to be easily stored or retrieved by just using the **MEMO** key. It can be used in many ways.

For example, it can be used as a telephone directory, time table, schedule, chart, etc.

Also, since retrieval, access, and write-in can be performed in a BASIC program, the utilization range can be expanded such as for a customer list, product list, estimated calculation, catalogue of books, etc.

There are many different ways to utilize the DATA BANK function in addition to the above items.

For details, See "CHAPTER 4 DATA BANK FUNCTION".

2-3. SIMPLE CALCULATION, AT THE BEGINNING

Simple calculation is performed as follows. However, if you have never used a scientific calculator, please be careful because this computer is provided with True Algebraic Logic functions in which multiplication and division are performed before addition and subtraction.

Example 1) $23 + 4.5 - 53 = -25.5$

Operation) 23 **+** 4.5 **-** 53 **EXE**

-25.5

* Numeral keys are shown without a frame from now on.

Example 2) $56 \times (-12) \div (-2.5) = 268.8$

Operation) 56 **x** 12 **-** 2.5 **EXE**

268.8

* To enter a negative number, press the **-** key before a number.

Example 3) $7 \times 8 - 4 \times 5 = 36$

Operation) 7 **x** 8 **-** 4 **x** 5 **EXE**

36

* Multiplication is performed first, then subtraction is performed.

Example 4) $(4.5 \times 10^{75}) \times (-2.3 \times 10^{-78}) = -0.01035$

Operation) 4.5 **E** 75 **x** 2.3 **E** 78 **EXE**

-0.01035

* Press the **E** key then enter the exponent.

There is another algebraic calculation which uses the memory. This memory is convenient when a certain numerical value is calculated in many different ways.

For example,

$$\begin{aligned} 3x + 5 &= \\ 4x + 6 &= \\ 5x + 7 &= \end{aligned}$$

When the value of x is 123.456 in these calculations, it is troublesome to press the same numerical value repeatedly. Is there any way to perform these calculations without this trouble? The solution is to use a memory called a variable. In these examples, since x is used for algebraic calculations, the calculations are performed by using variable X .

First, assign 123.456 to variable X.

$\boxed{X} = 123.456 \text{ EXE}$

" $\boxed{=}$ " does not mean equal but means that 123.456 is assigned to variable X. Let's perform these calculations.

$3 \times \boxed{X} + 5 \text{ EXE}$

$4 \times \boxed{X} + 6 \text{ EXE}$

$5 \times \boxed{X} + 7 \text{ EXE}$

375.368
499.824
624.28

They can be performed easily.

Since this computer is provided with 26 variables from A to Z, many different numerical values can be memorized.

In these examples, the numerical value for variable X is fixed and the calculation expressions are different.

However, how about a case in which the calculation expressions are fixed and the values of the variables are different?

When a calculation expression is determined to be " $3x + 5 =$ " and the value of x is changed to 123, 456, and 789, the operation is troublesome if the procedure mentioned above is used. Actually, the calculation expression is memorized by the computer and it is only necessary to change the value of variable X. This convenient calculation method is called "program calculation". A strong point of this computer is program calculation. Manual calculation, a previous step in which a program is used, is performed here. See "CHAPTER 3 BASIC PROGRAMMING" for a Program.

2-4. FUNCTION CALCULATION — A HIGHLIGHT OF THIS COMPUTER

This computer is provided with scientific functions as well the four basic functions.

Although these functions can be utilized in a program, manual utilization is explained here.

The functions provided by this computer are as follows.

Name of functions		Format
Trigonometric functions	$\sin x$	SIN x
	$\cos x$	COS x
	$\tan x$	TAN x
	$\sin^{-1} x$	ASN x
Inverse trigonometric functions	$\cos^{-1} x$	ACS x
	$\tan^{-1} x$	ATN x
	$\sqrt{x}$	SQR x
	$\log x$	LOG x
Square root	$\ln x$	LN x
Common logarithm	e^x	EXP x
Natural logarithm	x^y	$x \uparrow y$
Exponential function		DMS $\$ (x)^*$
Power		DEG $(x,y,z)^*$
Decimal $\rightarrow$ sexagesimal		INT x
Sexagesimal $\rightarrow$ decimal		FRAC x
Integer		ABS x
Integer removal		SGN x
Absolute value		
Coding	$ x $ Positive No. $\rightarrow 1$ $0 \rightarrow 0$ Negative No. $\rightarrow -1$	
	x is rounded off at the 10^y position.	RND $(x,y)^*$
Rounding off		
		RAN #
Random number generation		

* For DMS \$, DEG, and RND, the argument must be inside ().

Perform calculations with functions.

• Trigonometric functions (sin, cos, tan), and inverse trigonometric functions ($\sin^{-1}$, $\cos^{-1}$, $\tan^{-1}$)

When trigonometric functions and inverse trigonometric functions are used, always be sure to specify the angle unit (DEG, RAD, GRA).

Example) $\sin 12.3456^\circ = 0.2138079201$

Operation) MODE $\rightarrow$ "DEG"

SIN 12.3456 EXE

0.2138079201

* From now on, alphabet keys are shown without frames.

Example) $\cos 63^\circ 52' 41'' = 0.4402830847$

Operation) COS DEG $\rightarrow$ 63 $\rightarrow$ 52 $\rightarrow$ 41 $\rightarrow$ EXE

0.4402830847

Example) $2 \cdot \sin 45^\circ \times \cos 65.1^\circ = 0.5954345575$

Operation) 2 $\rightarrow$ SIN 45 $\rightarrow$ COS 65.1 EXE

0.5954345575

Example) $\sin^{-1} 0.5 = 30^\circ$

Operation) ASN 0.5 EXE

30

Example) $\cos(\frac{\pi}{3} \text{ rad}) = 0.5$

Operation) MODE $\rightarrow$ "RAD"

COS $\rightarrow$ $\frac{\pi}{3}$ $\rightarrow$ 3 $\rightarrow$ EXE

0.5

Example) $\cos^{-1} \frac{\sqrt{2}}{2} = 0.7853981634 \text{ rad}$

Operation) ACS $\rightarrow$ SQR 2 $\rightarrow$ 2 $\rightarrow$ EXE

0.7853981634

Example) $\tan(-35 \text{ gra}) = -0.612800788$

Operation) MODE $\rightarrow$ "GRA"

TAN $\rightarrow$ 35 EXE

-0.612800788

• Logarithmic functions (log, ln), and exponential functions (e^x , x^y)

Example) $\log 1.23 (= \log_{10} 1.23) = 0.0899051114$

Operation) LOG 1.23 EXE

0.0899051114

Example) $\ln 90 (= \log_e 90) = 4.49980967$

Operation) LN 90 EXE

4.49980967

Example) $e^5 = 148.4131591$

Operation) EXP 5 EXE

148.4131591

Example) $10^{1.23} = 16.98243652$

(The anti-logarithm of common logarithm 1.23 is obtained.)

Operation) 10 $\rightarrow$ 1.23 EXE

16.98243652

Example) $5.6^{2.3} = 52.58143837$

Operation) 5.6 $\rightarrow$ 2.3 EXE

52.58143837

Example) $123^{1/7} (= \sqrt[7]{123}) = 1.988647795$

Operation) 123 $\rightarrow$ 1 $\rightarrow$ 7 $\rightarrow$ EXE

1.988647795

* When $x < 0$, y is a natural number.

Example) $\log \sin 40^\circ + \log \cos 35^\circ = -0.278567983$

The anti-logarithm is 0.5265407845 (logarithmic calculation of $\sin 40^\circ \times \cos 35^\circ$).

Operation) MODE $\rightarrow$ "DEG"
LOG SIN 40 $\rightarrow$ LOG COS 35 EXE
10 $\rightarrow$ EXE

-0.278567983
0.5265407845

• Other functions ($\sqrt{\quad}$, SGN, RAN#, RND, ABS, INT, FRAC)

Example) $\sqrt{2} + \sqrt{5} = 3.65028154$

Operation) SQR 2 $\rightarrow$ SQR 5 EXE

3.65028154

Example) Gives "1" if it is a positive number, "-1" if it is a negative number, and 0 if it is "0".

Operation) SGN 6 **EXE**
SGN 0 **EXE**
SGN -2 **EXE**

1
0
-1

Example) Random number generation (Pseudo random numbers with the range of $0 < \text{RAN} \# < 1$).

Operation) RAN **SHIFT** **EXE**

0.7903739076

(Random number)

Example) Round off the result of 12.3×4.56 to one decimal place.

$$12.3 \times 4.56 = 56.088$$

Operation) RND **SHIFT** 12.3 **EXE** 4.56 **EXE** 2 **SHIFT** **EXE**

* When RND (x,y) is used,
 $|y| < 100$.

56.1

Example) $|-78.9 \div 5.6| = 14.08928571$

Operation) ABS **SHIFT** -78.9 **EXE** 5.6 **SHIFT** **EXE**

14.08928571

Example) Integer of $7800/96$ 81

Operation) INT **SHIFT** 7800 **EXE** 96 **SHIFT** **EXE**

81

* The maximum integer that does not exceed the original numerical value is obtained by this function.

Example) The fraction of $7800/96$ 0.25

Operation) FRAC **SHIFT** 7800 **EXE** 96 **SHIFT** **EXE**

0.25

- **Designation of number of significant positions, and designation of number of decimal positions.**

The number of positions are designated by the "SET" command.

Designation of number of significant positions SET E n ($n = 0$ to 8)

Designation of number of decimal positions SET F n ($n = 0$ to 9)

Cancellation of designations of number of positions ... SET N

* In a manual calculation, "SET E0" designates an 8 significant positions.
* Even if a designation is performed, the original numerical value remains in the memory.

Example) $100 \div 6 = 16.66666666$

Operation) SET E 4 **EXE** (Designation of 4 significant positions.)

100 **EXE** 6 **EXE**

16.67

Example) $123 \div 7 = 17.57142857$

Operation) SET F 2 **EXE** (Designation of 2 decimal positions)

123 **EXE** 7 **EXE**

17.57

Example) $1 \div 3 = 0.33333333$

Operation) SET N **EXE** (Designation cancellation)

1 **EXE** 3 **EXE**

0.33333333

- **Decimal $\leftrightarrow$ sexagesimal conversion (DEG, DMS \$)**

Example) $14^{\circ}25'36'' = 14.42666667$

Operation) DEG **SHIFT** 14 **EXE** 25 **EXE** 36 **SHIFT** **EXE**

14.42666667

Example) $12.3456^\circ = 12^\circ 20' 44.16''$

Operation) DMS $\square$ $\square$ 12.3456 $\square$ $\square$ EXE 12° 20' 44.16

Example) $\sin 63^\circ 52' 41'' = 0.897859012$

Operation) MODE $\square$ 4 $\square$ SIN DEG $\square$ $\square$ 63 $\square$ 52 $\square$ 41 $\square$ $\square$ EXE 0.897859012

CHAPTER 3

“BASIC” PROGRAMMING

In this chapter, BASIC programs and programming are explained. Users who are not accustomed to programming should read this chapter to master the basics of programming.

3-1. WHAT IS A PROGRAM?

The word "PROGRAM" may sound like something difficult. However, there are very simple programs and more complicated ones. For example, calculation, in which algebraic expressions are memorized and numerical values are assigned to these expressions, is also a program.

3-1-1 A Program Is Convenient

We are concerned with many different kinds of calculations such as those for financial business accounting, measurements, or for housekeeping and expenses. Although it is not so troublesome if these calculations are performed only once, it is tedious to perform calculations repeatedly with the same calculation expression while changing numerical values. Because of this, these calculations can be best performed by using your computer. For example, if the calculation expression $y=2x^2+5x+13$ is used, to obtain the value of y when the value of x is changed, the same calculation must be repeated. To eliminate this trouble, the following expression is placed in memory.

```
10 INPUT X
20 Y=2*X^2+5*X+13
30 PRINT Y
```

In this program a calculation expression is memorized. A detailed explanation will be provided later. This program allows the calculation to be easily performed.

Simple programs can be conveniently used just by memorizing a calculation expression as mentioned above.

Next, the program will be sequentially explained.

3-1-2 Program Construction

Remember program construction.

```
10 INPUT X
20 Y=2*X^2+5*X+13
30 PRINT Y
```

This program can be divided into 3 parts as follows.

```
10 INPUT X ..... Input
20 Y=2*X^2+5*X+13 ..... Calculation
30 PRINT Y ..... Output
```

At first, the input part is used to enter (input) data (such as numerical values for calculation) into the computer. Next, the calculation part is used to perform a calculation so that an answer can be provided. Last, the output part is used to provide (display) an answer. A computer does everything required if correct commands (instructions) are provided. In this example, an input command (INPUT) and an output command (PRINT) are memorized. These three parts can be further broken down as follows.

Line No.	Command	Operand
10	INPUT	X

The line numbers indicate the sequence of the program flow. Since a computer reads and executes statements in ascending order of line numbers, place these line numbers according to the expected execution sequence. Also, it is advisable to assign these numbers in 10s (10, 20, 30, ...) because this is convenient if additions are required later. Decimal figures such as 1.5 or 12.3 cannot be used for line numbers.

The items that follow the line number are the commands to be performed by the computer. There are many different kinds of commands used for specification of instruction required.

Although it is desirable to remember all the commands, just memorize the minimum necessary commands at first, then the rest gradually. See "CHAPTER 5 COMMAND REFERENCE" on page 141 and after for the kinds of commands and their functions. The function of an operand next to a command is to supplement it. Some commands have an operand while other commands do not. In this example, the INPUT command indicates the entry of data. An operand specifies memory where entered data is placed; in this case, the entry to variable X.

In the following line,

20 Y=2*X+2+5*X+13
 Line No. Assignment statement

20 is the line number. The assignment statement means that the value on the right of the equal sign (=) is entered (assigned) to the variable on the left. If the LET command is added to the assignment as follows,

20 LET Y=2*X+2+5*X+13

LET is a command and the assignment statement is an operand. Line 30 consists of a line number, command, and an operand the same as line 10.

30 PRINT Y
 Line No. Command Operand

Program construction is as mentioned above. Additional commands, operands, and line numbers are used to construct a large program.

3-1-3 Easy Program Preparation

When program construction is understood, it is not so difficult to prepare a program.

After the three main parts (input, calculation, and output) are understood, they can be used to prepare a program. It is unnecessary to remember all the commands at one time. It is advisable to try simple calculation by just using "INPUT", "PRINT", and an assignment statement.

The secret of quick program mastery is not to just remember a program, but to actually prepare a program by selecting a problem that can be easily placed into a calculation expression from among those around you such as the calculation of bills or financial calculation repeatedly performed in a company, measurements, housekeeping and time calculation for cassette recording often performed at home. The best way to master programming is to first prepare a program for the subject to process, memorize the required commands, then improve the program.

Another secret is not to prepare the entire program at one time but to prepare its different parts and put them together later.

Prepare a program gradually and slowly by using this concept. After fundamental program construction has been understood, a program can be prepared by referring to each command function in the "CHAPTER 5 COMMAND REFERENCE" and by actually using them.

3-2. PROGRAM PREPARATION

This section and after cover the main subject, BASIC Programming, in which a program is actually prepared.

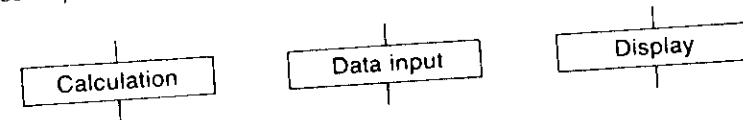
3-2-1 Preparing A Flowchart

You may not be accustomed to a flowchart. It is a chart which describes a work sequence.

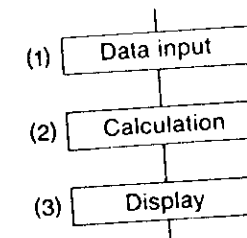
It is often heard that a flowchart is not required for BASIC; this is correct for a user who is accustomed to utilizing a computer. However, since the entire work procedure is hard to understand for a beginner, it is important to draw a simple flowchart to understand the program flow.

While there are formal and dedicated symbols for drawing flowcharts, it is unnecessary to remember these symbols. Just place each work item inside and connect them with lines.

Let's prepare a program to explain each item. For example, make a program to obtain the square of a numeral by entering it. First, a calculation part is required. Since a numeral is entered, an input part is necessary, and since an answer is displayed, an output part is necessary. The three parts are placed inside as follows.



When these three items are sequentially connected, it is easily found that the last item is the display of the answer. Next, it is necessary to perform calculation to obtain an answer, and also to enter data to perform calculation. Connect these three items.

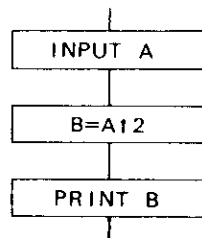


As the flowchart has been completed by this procedure, let's change it into a format that is more like a program.

The first item is an instruction to enter data. Since data is entered into a variable by using an INPUT statement, determine the variable. If variable A is used, the content of (1) is "INPUT A".

In the calculation performed in the second item, the entered content of variable A is squared, and the answer is assigned to another variable. If this variable is B, then "B=A²". This calculation expression is called an assignment statement which is formally written as "LET B=A²". However, since LET can be omitted, it can be written as "B=A²".

The answer is displayed by the third item. Since the content of variable B which includes the answer is displayed by using a PRINT statement, it is written as "PRINT B". These three items are placed into a flowchart again.



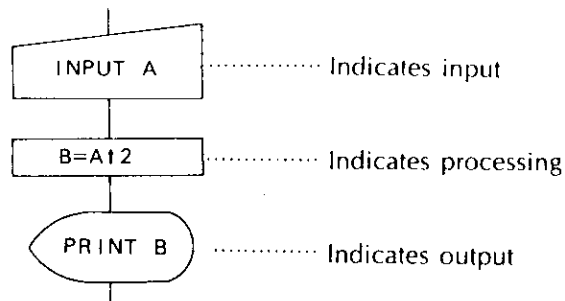
This program is completed by placing line numbers for these three items.

```

10 INPUT A
20 B=A^2
30 PRINT B
  
```

A program can be quickly and easily made by sequentially assembling a flowchart after preparing each item as mentioned above.

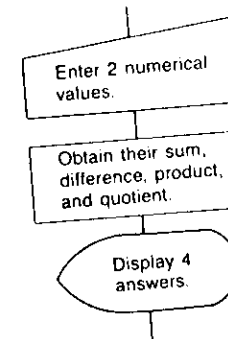
An actual flowchart has formal symbols which have special meanings. Draw the above example by using them.



Refer to the flowchart symbol at the end of this manual.

3-2-2 Preparing A Program

To provide a simple example, enter 2 numerical values and obtain their sum, difference, product, and quotient. Prepare a flowchart based on the input, calculation, and output parts which are the three important items.

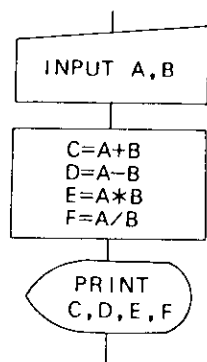


Use an INPUT statement (an instruction to enter data from the keyboard into a variable) to enter 2 numerical values. Items that should be noted are the variables where data or an answer are entered. The utilization of variables is quite complicated. Variables include those with alphabetical characters from A to Z, and array variables that have an item called a subscript such as A(3). Although a variable can be selected from among these variables, it is recommended that variables be selected in alphabetical order while you are not accustomed to programming. Two numerical values are entered here. A and B are selected and "INPUT A, B" is written. Several variables can be handled in one INPUT statement by punctuating them with commas. How about the calculation parts? Four items are calculated here; four answers will be obtained. The variables where the four answers are entered will be C, D, E and F.

First, For addition of A+B, C=A+B is used.
 For subtraction of A-B, D=A-B is used.
 For multiplication of A*B, E=A*B is used.
 For division of A/B, F=A/B is used.

Since this completes the calculation items, the answers are displayed next. The display command is PRINT; "PRINT C, D, E, F" is realized.

The program format can be placed in a flowchart as follows.



Complete the program by placing line numbers.

```

10 INPUT A, B
20 C=A+B
30 D=A-B
40 E=A*B
50 F=A/B
60 PRINT C, D, E, F
  
```

Next add "END" to indicate program termination.

```

70 END
  
```

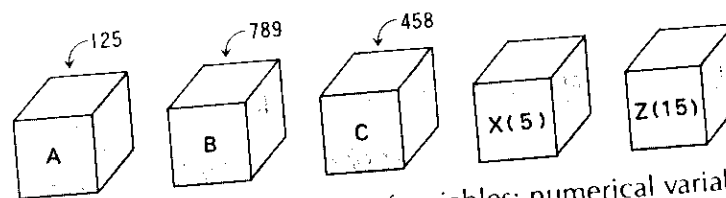
The program has been completed by the above procedure. It can be easily written if it is assembled sequentially.

First, prepare a simple and practical program instead of complicated one by using many different commands.

NOTE

VARIABLES

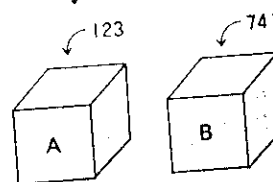
Variables are important elements for program preparation. Variables are just like boxes where entered data or calculated data are stored with each having a name. Variables include the standard ones from A to Z and those with a subscript attached to the name (A to Z). The latter ones are called array variables such as A(5) and B(50).



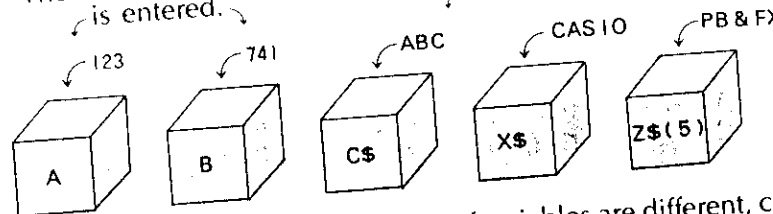
Also, there are two different types of variables; numerical variables where numerical values are entered, and character variables where character strings are entered. The variables that were previously used are numerical variables where numerical values have been entered to perform a calculation. In addition to these, there are character variables with \$ attached to the name (A to Z), such as A\$, B\$, C\$, and a special character variable called "exclusive character variable", \$.

A numerical value with up to 10 digits (10 digits in the mantissa part, 2 digits in the exponent) can be entered into a numerical variable, while a string with up to 7 characters can be entered into a character variable. Also, up to 62 characters can be entered into the exclusive character variable.

Numerical variables
The numerical value is entered.



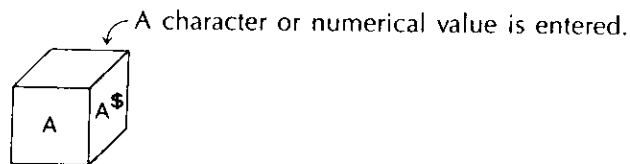
Character variables
The characters are entered.



Since the items entered in these two kinds of variables are different, characters such as "ABC" cannot be entered into numerical variables while numerical values for calculations cannot be entered into character variables. The utilizations of these variables are different. Use numerical variables when numerical values are to be entered for calculation, and character variables when messages or symbols are to be entered. Arrays are convenient when data are stored in many variables. They are distinguished by subscripts indicating the 1st variable, 2nd variable, etc. Array variables will be explained by utilizing them in a program.

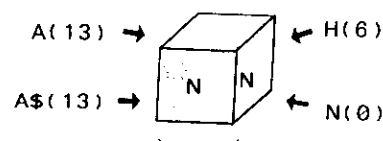
<Precautions when using variables>

If a numerical variable has the same name as a character variable when a program is used, data will be entered into the same place.



As a result, the numerical variable A and the character variable A\$ cannot be used simultaneously in the same program.

Also, when an array variable is used (see page 77) precautions must be taken so as not to give several equivalent names to one container.



All names are for the same container.

A=A(0)=A\$=A\$(0)

B=A(1)=B(0)=B\$=A\$(1)=B\$(0)

C=A(2)=B(1)=C(0)=C\$=A\$(2)=B\$(1)=C\$(0)

⋮

N=A(13)=B(12)=.....=N(0)=N\$=A\$(13)=.....N\$(0)

⋮

Z=A(25)=B(24)=C(23)=.....=Z\$=A\$(25)=.....Z\$(0)

Same precautions must be taken when memory is expanded by a DEFM statement.

3-2-3 Program Input

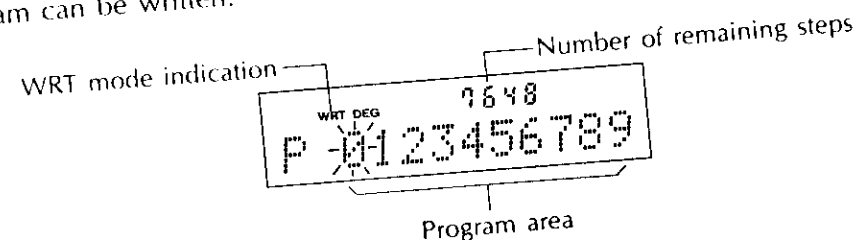
Memorize (input) a program.

Use the previous program in which four basic calculations are performed after entering two numerical values.

Program

```
10 INPUT A,B
20 C=A+B
30 D=A-B
40 E=A*B
50 F=A/B
60 PRINT C,D,E,F
70 END
```

When the power switch is turned on, the RUN mode, in which manual calculations or program execution can be performed, is specified. Press **MODE** **1** to switch from this mode to the WRT mode in which a program can be written.



This display shows a status in which no program is stored. The above 4-digit numeral indicates the number of remaining steps. This number decreases when a program is stored or when the memory is expanded (see page 80). Numerals from 0 to 9 are program area numbers; the blinking one indicates the currently specified program area. When a program is stored during this status, it is stored in program area P0. Different programs can be written in 10 program areas from P0 to P9. If a program is stored, the program area number is not displayed but the cursor, " _ ", is displayed. To erase all the programs and store a program in program area P0, enter

NEW ALL **EXE**

This is a command that erases all programs. Store a program with the following procedure.

10 INPUT A,B EXE — Press this key at the end of each line.

— This operation can also be performed by pressing (INPUT).

20 C=A+B EXE

30 D=A-B EXE

40 E=A*B EXE

50 F=A/B EXE

60 PRINT C,D,E,F EXE

70 END EXE

After the EXE key is pressed, a one character space is made after the line number to allow the display to be easily read.

Was the program correctly stored?

Press the keys slowly and firmly even if it is boring. When a wrong key was pressed, a correction can be made by the following operation.

• A mistake was noticed before the EXE key was pressed.

If this occurs, place the cursor at the location where the mistake occurred by using the (←) (→) keys and correct it.

Example 1) 10 INPUT S_ "S" was pressed instead of "A".

Place the cursor under the "S" by pressing the (←) key once.

(←) 10 INPUT S

Press the correct key.

(A) 10 INPUT A_

Complete the entry then press the EXE key.

(A) (B) EXE 10 INPUT A,B

Example 2) 40 EE=A*B_ An extra "E" was entered.

Press the (←) key 5 times and place the cursor under the second "E".

(←)(←)(←)(←)(←) 40 EE=A*B

Since 1 character is to be deleted, press the (DEL) key once.

(DEL) 40 E=A*B

After the deletion is made, press the EXE key.

40 E=A*B EXE

Example 3) RINT C,,E,F_ "D" on line 60 was skipped.
Press the (←) key 4 times to place the cursor next to the insertion location.

(←)(←)(←)(←)

Provide one character space.

(←) INS

Insert "D".

(D)

After the correction has been completed, press the EXE key.

EXE

• A mistake was noticed after the EXE key was pressed.
Since a line is stored as part of a program after the EXE key is pressed, recall it by using the LIST command to correct.

Example) Line 50 was mistakenly entered as "50 F=A/N".
Recall line 50 by using the LIST command.

(LIST) 50 EXE

Pressing (LIST) provides the same result.

Press the (←) key once to place the cursor under the "N".

(←)

Press the correct key.

(B)

After the correction has been completed, press the EXE key.

EXE

If lines 60 and after are stored, line 60 is displayed.

If no other correction is required, press the (AC) key to clear the display.

(AC)

After the EXE key is pressed, a correction can be performed by recalling the line with the LIST command. Also, the line can be rewritten with a new line number.

When a new line is stored with a number which has been already used, the line stored later has priority, and the old one is erased.

A program is stored as mentioned above. After the storage operation has been completed, press **MODE**  to return to the RUN mode. If the WRT mode is maintained, a stored program may be erased or changed by mistake. Therefore, be sure to return to the RUN mode after completion of the storage operation.

NOTE

PROGRAM AREAS

This computer is provided with 10 program areas, P0 to P9 where independent programs can be stored. All these program areas can be used in the same way. For example, if these areas were not provided and if 3 programs were to be used very often, they would have to be loaded from tape each time. This computer can store these 3 programs in 3 program areas such as P0, P1 and P2.

Although this function is very convenient, precautions have to be taken concerning the number of steps used; the total number of steps used in all program areas must not exceed the maximum capacity.

A program area can be specified by pressing a key from  to  after pressing the **SHIFT** key. This specification can be done both in the RUN mode and WRT mode. In the RUN mode, the program stored in the specified area automatically starts. In the WRT mode, the program does not start but a program area, where a program is to be input or editing is to be performed, is specified.

The program areas must be correctly handled. When a program is executed, stored on cassette tape, or loaded from cassette tape, if a wrong area is specified, the operation cannot be performed correctly.

When the power is turned on, program area P0 is specified automatically. This can be confirmed by the numeral following "READY" after you press **MODE** .

Example) **MODE**  → READY P3 Program area P3

3-2-4 Program Execution

Perform calculation with the program that was previously stored. Press **MODE**  and confirm that the computer is in the RUN mode ("RUN" is displayed).

There are two different ways to execute a program.

- (1) Execution by specifying the program area.
After pressing the **SHIFT** key, press the numeral key from  to  which specifies a program area. If a program is stored, it will start.

Example) **SHIFT** **P0**

- (2) Execution by the RUN command.

Example) **SHIFT** **RUN** (same as **RUN**) **EXE**

The difference in these two execution methods is as follows. In method (1), execution always starts from the beginning of the program area while in method (2), execution can be started from the beginning or from an arbitrary line number.

Execute a program with method (1).

Operation

SHIFT **P0**

Enter 2 data.

Example)

45 **EXE**

36 **EXE**

Display

?

?

81

STOP

When the PRINT statement is executed, "STOP" is lit.

After 2 data were entered, the sum was displayed. Press the **EXE** key to display the next answer.

EXE

EXE

EXE

9

1620

1.25

STOP

STOP

STOP

Next, execute a program by using the RUN command. If RUN **EXE** is entered, the result is the same as that obtained by method (1). Therefore, execute it from line 20.

Operation

SHIFT RUN 20 **EXE**

(Same as RUN 20 **EXE**.)

EXE

EXE

EXE

81

9

1620

1.25

If no line number is specified when the RUN command is used, program execution starts from the beginning, and if a line number is specified, execution starts from that line number as mentioned above.

Actually there is another difference between these two methods.

When the RUN command is used, the program in the currently specified program area is executed. However, if P0 is to be executed while P5 is specified, what is the procedure?

The solution is to press **SHIFT** P0.

After a program has been prepared and stored, execute it. Even if an error (ERR is displayed) occurs after execution, don't be disappointed. In this case, find the cause of the error (debug) by referring to the following section.

NOTE**HOW TO COUNT THE NUMBER OF STEPS**

This computer is provided with a memory capacity of 7648 steps.

A step is the unit that indicates the memory capacity in which a program can be stored. As a program is stored, the number of remaining steps is reduced.

When the WRT mode is specified by pressing **MODE** (1), the current number of remaining steps is displayed.

Example)

EXE (1)

WRT 7648
P 0123456789

Number of remaining steps

The number of steps is counted as follows.

- Line No..... 2 steps per line No. from 1 to 9999.
- Command..... 1 step
- Function..... 1 step
- Character..... 1 step per character.
- In addition, each press of the **EXE** key during storage is counted as 1 step.

Example)

1 INPUT A **EXE** 5 steps
 2 1 1
 10 B=SIN A **EXE** 7 steps
 2 1 1 1 1
 100 PRINT "B=" ; B **EXE** 10 steps
 2 1 4 1 1 1 Total: 22 steps

- When the memory is expanded, 8 steps are required for 1 memory expansion.

Example) Initial state.....26 memories 7648 steps
 DEFM 10 **EXE** 36 memories 7568 steps

3-2-5 Debugging (error correction)

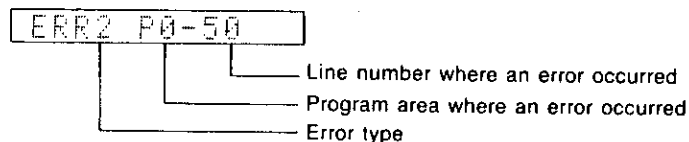
After a program has been prepared and executed, it often happens that an error is displayed and a result cannot therefore be obtained. Don't be disappointed since the cause of this error can be found.

Eliminating errors is called "debugging".

The debugging method depends on the cause of the errors. In some cases, an error is displayed during execution, and in other cases, an error is not displayed but a result cannot be obtained as it is supposed to be. When an error is displayed during execution, its location and its type are shown. As a result, the cause can be easily found. However, when a correct result is not obtained without displaying any error, this is troublesome.

(1) Debugging with the error display.

The error display provides "error message" which indicates the following three items.



The error type is indicated by a code number that follows "ERR". The code number from 1 to 9 is used to indicate type; "ERR1" indicates "memory overflow", and "ERR2" indicates "syntax error". See the "Error Message Table" on page 188 for the meaning of these code numbers.

The program area and line number where the error occurred are also indicated.

Where and what kind of error occurred can therefore be determined by these three items.

Let's take a look at an example.

An error that often occurs is "ERR2" which is a syntax error. It occurs when a program is incorrectly stored.

For example, the program used in the previous example is incorrectly stored.

Operation

MODE 1
SHIFT LIST 20 EXE
[C] [C] DEL
EXE
AC MODE [C]

Display

P	_123456789
20	C=A+B_
20	C=AB
30	C=A-B_
READY P0	

In this example, "C=A+B" on line 20 was entered as "C=AB" by mistake. Now, execute the program.

Operation

SHIFT P0
45 EXE
12 EXE

Display

?	
?	
ERR2 P0-20	

An error message is displayed; it indicates that a syntax error occurred on line 20 in program area P0.

Check line 20.

Operation

AC.....Error release
MODE 1
SHIFT LIST 20 EXE

Display

P	_123456789
20	C=AB_

Check if the line is correctly written. Since "+" between A and B was left out, correct this.

Operation

[C]
SHIFT INS
+ EXE
AC MODE [C]

Display

20	C=AB
20	C=A_B
30	C=A-B
READY P0	

Since "ERR2" is mostly caused by erroneous program input, when "ERR2" occurs, check the line whose number is indicated in the error message. When data is read-in by a READ statement (see page 82), if character data is read into a numerical variable, "ERR2" is also displayed. When a READ statement exists at an "ERR2" location, check the data in the DATA statement, too.

Check points for various errors are as follows.

- ERR1: Shortage of memory. Stack over.
Confirm the number of remaining steps. Check if the memory has been mistakenly expanded beyond the capacity by a DEFM statement. Check if the calculation expressions are too complicated.
- ERR2: Syntax error
Check if there are any errors in the stored program.
- ERR3: Mathematical error
Check if the arithmetic result of a calculation expression is more than 10^{99} , or if the input range of a function is exceeded. When variables are used, check their contents.
- ERR4: Undefined line number
The specification of a line number in a GOTO, GOSUB, or RESTORE statement is not correct. Confirm this line number.
- ERR5: Argument error
Check the value of an argument or parameter for a command or function. When variables are used, check their contents.
- ERR6: Variable error
When an array variable is used, check if the memory is expanded by a DEFM statement. Also, check if the same variable is used for both character variables and numerical variables at the same time.
- ERR7: Nesting error
If the line where an error occurred is a RETURN statement or NEXT statement, check if the GOSUB statement or FOR statement correspondence is correct or not. Also, if the line where the error occurred is a GOSUB statement or FOR statement, check if there are more than 8 nesting levels for a GOSUB statement, and more than 4 for a FOR statement.
- ERR8: Password error
When a password is specified, check if another password was entered, or if LIST, NEW, NEW ALL, etc. were used.

- ERR9: Option error
Check if the FA-3, FA-5 cassette interface, SB-42 interface pack for FP-40 character printer or FP-12S character mini-printer are connected properly or not. Check if the rechargeable battery of the FP-12S is charged with paper. Also, adjust the volume or tone of the tape recorder connected to the FA-3, clean the tape recorder head and replace the tape with a new one. Or operate the tape recorder by using only the white plug when recording is performed, and only the black plug during tape playback. The commands mentioned above will be explained later. See "CHAPTER 5 COMMAND REFERENCE" on page 141 and after for details.

(2) Although to error is displayed, the desired result cannot be obtained.

Since this often happens when a calculation expression or variable in a program was incorrectly used, check the operation of calculation expressions and variables. Especially when a correct result is not obtained, compare the original expression with the expression used in the program. When the program execution does not stop or stops without work being accomplished, check the operation of the variable that controls the program flow. In regard to a calculation expression, check its location in the WRT mode (press **MODE** **1**).

The program flow can be checked by stopping it with a STOP statement after entering data into the control variable, or by displaying the value of a variable with a PRINT statement. Store the following program.

```

10 INPUT A
20 B=1
30 FOR C=1 TO A
40 B=B*C
50 C=C+1
60 NEXT C
70 PRINT B
80 END

```

This program obtains factorial of data entered by an INPUT statement. Actually line 50 is not required, and the variable used for FOR loop should not be changed.

The FOR-NEXT statements on lines 30 and 60 form a loop in which calculation is repeatedly performed. (These statements will be explained later.)

Operation

MODE 1

SHIFT P1

NEW EXE

Command to erase a program.

10 INPUT A EXE

20 B=1 EXE

30 FOR C=1 TO A EXE

40 B=B*C EXE

50 C=C+1 EXE

60 NEXT C EXE

70 PRINT B EXE

80 END EXE

Display

P	123456789
P	123456789
P	123456789

10	INPUT A
20	B=1
30	FOR C=1 TO
40	B=B*C
50	C=C+1
60	NEXT C
70	PRINT B
80	END

Press **AC** **MODE** **2** to specify the RUN mode.

Operation**Example)** SHIFT P1

12 EXE

Display

?
10395

The correct answer is 479001600.

Check the calculation expression. It has no mistake. Next, check the FOR-NEXT loop flow.

Insert a STOP statement after line 50 to stop the program each time.

Operation

MODE 1

55 STOP EXE

AC MODE 2

Display

P	123456789
55	STOP
READY	P1

Since a STOP statement is to be inserted after line 50, place a line number between 50 and 60. As line number, 55 was selected. Execute the program.

Operation

SHIFT P1

12 EXE

Check the value of loop control variable C.

C EXE

Continue execution.

EXE

Check the value of variable C again.

C EXE

Although the value of variable C must be increased by one each time, it is increased by 2. Therefore, it was found that the FOR-NEXT loop operation (flow) is not correct. Check the increment of variable C again. It was found that the problem is line 50 which is not required. Therefore, delete line 50 and the STOP statement that was added on line 55.

Operation

MODE 1

50 EXE

55 EXE

AC MODE 2

Display

P	123456789

READY	P1

Debugging has been completed.

There is another way to debug besides using the STOP statement. It is debugging in the trace mode (Press **MODE** **2**. "TR" is displayed.) In regard to debugging in the trace mode, a program is executed and stopped after each command. Since the value of a variable, etc. is checked while the program is stopped, debugging is performed by pressing the **EXE** key to advance to the next command.

Try this with the previous example. The program area and line number are displayed each time the **EXE** key is pressed. Press **MODE** **3** to release the trace mode; "TR" is erased.

Since debugging can be performed as mentioned above, when an error is displayed or when the desired result cannot be obtained, don't be disappointed but try debugging.

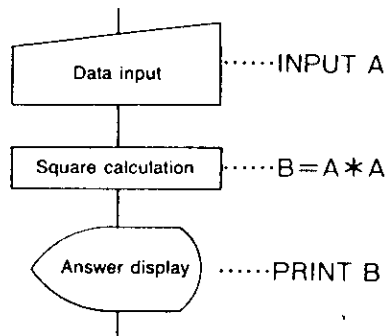
3-3. PROGRAM DEVELOPMENT

It is certain that the outline of program has been understood by the previous explanation. The three parts of a program are input, calculation, and output. Many different programs can be prepared with these three items. However, a program can be more convenient and easier by using the commands explained in this section.

3-3-1 Changing The Program Flow (GOTO statements)

In addition to the three parts of a program, GOTO statements are very convenient when the same calculation must be repeated many times, or to transfer program flow to an arbitrary line instead of following line numbers sequence.

For example, let's prepare a program to obtain the square of a certain value. This program can be broken down in three parts which are "data input", "square calculation", and the "answer display". Make a flowchart.



Prepare the program in accordance with this flowchart.

```
10 INPUT A
20 B=A*A
30 PRINT B
40 END
```

For example, square 15 and 43.

Operation
RUN **EXE**
15 **EXE**
RUN **EXE**
45 **EXE**

Display
?
225
?
1849

Since execution has to be performed each time as mentioned above, it is very inconvenient when lots of data exist. Do you think it is very convenient that calculation can be repeatedly performed? It is the GOTO statement that makes this possible. The function of a GOTO statement is to transfer program flow to a line number or program area specified by the numerical value following GOTO. Replace the END statement on line 40 with a GOTO statement.

40 GOTO 10

This means that the flow after line 40 is transferred (jumped) to line 10. Execute the modified program.

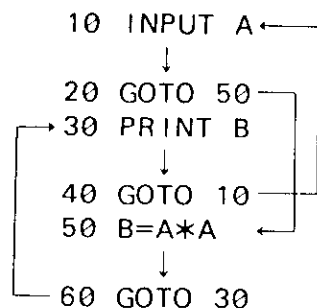
Operation
RUN **EXE**
15 **EXE**
EXE
43 **EXE**

Display
?
225
?
1849

A GOTO statement is convenient for the repetition of calculation as mentioned above. Since a GOTO statement can cause a return to the beginning of a program to repeat execution, and can also cause a jump to an arbitrary location, there are many convenient ways to use it. For example,

```
10 INPUT A
20 GOTO 50
30 PRINT B
40 GOTO 10
50 B=A*A
60 GOTO 30
```

The flow of this program is as follows.



Since a GOTO statement unconditionally causes a jump to a specified line number as shown above, it is called an "unconditional jump."

A jump to a program area as well as to a line number can be performed by a GOTO statement. The program area is specified by adding "#" and a number from 0 to 9.

Example) GOTO #1 Jumps to program area P1.

GOTO #9 Jumps to program area P9.

When a jump is made to a program area, execution continues from the beginning of the program in this area.

NOTE

PRINT STATEMENTS

A PRINT statement is used for displaying the content of a variable, character string, or numerical value. Numerical variables and character variables can both be used.

Example) When A=123 PRINT A → 123

When B\$="ABC" PRINT B\$ → ABC

Since a character string placed inside "" (quotation marks) is displayed as it is, it can be used as a message.

Example) PRINT "CASIO" → CASIO

When two or more items are to be displayed, they can be written by punctuating them with commas (,) or semicolons (;).

Example) PRINT A,B,Z\$
PRINT "TOTAL=";T

Note that when a "," is used, output is performed with line change; the execution stops after the first content is displayed ("STOP" appears). The following display is obtained by pressing the **EXE** key. However, when a ";" is used, continuous display is performed.

Example) Try this by using the following program.

```

10 A=123
20 B$="ABC"
30 PRINT A,B$ ..... After displaying the content of A, the
                      content of B$ is displayed by pressing
                      the EXE key.

40 PRINT A;B$ ..... The content of A and B$ are displayed
                      continuously.

50 PRINT B$: ..... After displaying the content of B$,
                      the execution advances.

60 PRINT A ..... After displaying the content of A, the
70 END ..... execution stops.
  
```

This program is executed as follows.

Operation

RUN **EXE**

EXE

EXE

EXE

Display

123	} PRINT A,B\$
ABC	
123ABC	} PRINT A;B\$
ABC 123	

A one character space exists before the numerical value (123). This is the space for a sign (+ or -); since the positive sign is always omitted, a space is opened.

3-3-2 Condition Test By A Program (IF-THEN statement)

If a size could be determined or control could be automatically performed in a program, it would be convenient.

An IF statement makes a test in a program; it makes a test on a conditional expression.

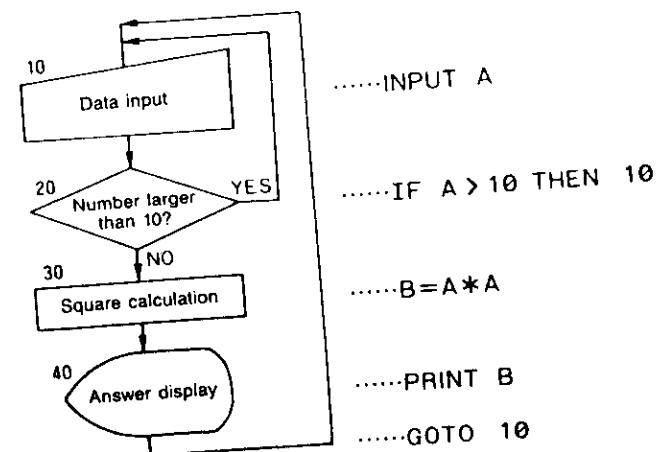
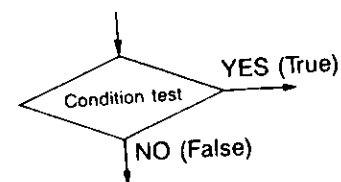
IF conditional expression THEN { Line No. or program area }
Instruction statement

If the conditional expression is true, a jump is made to the line number or program area following "THEN", or the statement following "THEN" is executed. If the conditional expression is false, program execution advances to the next line.

Let's check the function of an IF statement.

Example) Enter an arbitrary number. If it is larger than 10, a return is made to the data input status. If it is 10 or smaller, its square is calculated and displayed then a return to the data input status.

This program consists of 4 parts (Input, Condition test, Calculation and Display). The following symbol is used for a condition test flowchart.



The numerals at the left of the flowchart are line numbers.

```
10 INPUT A
20 IF A>10 THEN 10
30 B=A*A
40 PRINT B
50 GOTO 10
```

Line 20 uses an IF statement. If the condition is true, the item following THEN is executed. In this case, the program jumps to line 10.

The following relational operators are used for conditional expressions.

Left side > right side ... Left side is larger than the right side.
 Left side < right side ... Left side is smaller than the right side.
 Left side = right side ... Left side is equal to the right side.
 Left side $\geq$ right side ... Left side is larger than or equal to the right side.
 Left side $\leq$ right side ... Left side is smaller than or equal to the right side.
 Left side $\neq$ right side ... Left side is not equal to the right side.

Since "THEN" includes the meaning of "GOTO", "THEN GOTO 10" can be written as "THEN 10".

Execute this program.

Operation

RUN **EXE**

5 **EXE**

EXE

12 **EXE**

9 **EXE**

Display

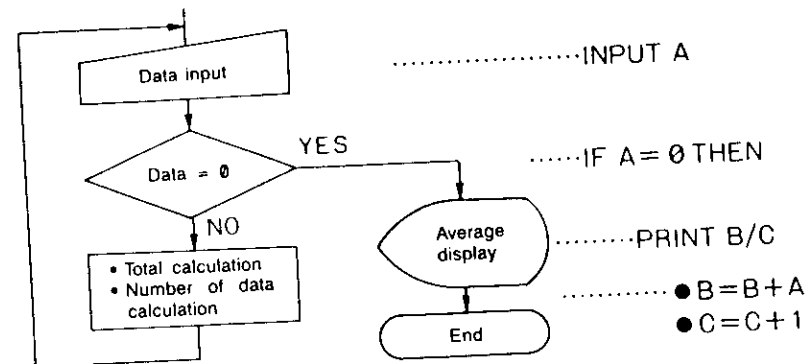
?
25
?
?
81

Data can be selected by an IF statement as mentioned above.

Example) When "0" is entered after entering several data, the average of these data is obtained.

This program can be divided into "Input", "Condition test", "Calculation" and "Display" parts. The "Calculation" part includes three procedures; obtaining the total, counting the number of items, and obtaining the average. Since the average calculation is only executed when "0" is entered, it will follow the "Condition test" part.

Prepare a flowchart based on this analysis.



As can be seen in this flowchart, the IF statement checks if data entered to variable A is 0. If it is not 0, the total and number of data are obtained after which a return is made to data input status. If it is 0, the average is displayed and the execution terminates. Note that if the first input is 0, a division by zero causes an error.

This example is slightly difficult compared to the previous one. In regard to the calculation part, the input data are added to the variable for the total calculation. Since B is the variable for the total, the calculation is "B=B+A". (The content of variable A is added to the content of variable B.)

In regard to the number of data, when a data is entered, 1 is added to the counting variable (C in this case); "C=C+1" is realized.

In regard to the condition test part, which is the main part, if A is 0, the average is displayed by a PRINT statement. Since a statement can be written following THEN of an IF statement, "PRINT B/C" is written in this example; the result of the calculation expression (B/C) is displayed.

In regard to variables B and C, if values were previously entered into them, they are continuously incremented and a wrong answer is obtained. As a result, zero must be entered into these variables ("B=0", "C=0"). Although separate line numbers can be used for these two assignment expressions, it will be easier to use a multistatement (written as "B=0:C=0" on one line with punctuation by a ":" (colon)). Prepare the program.

```
10 B=0:C=0
20 INPUT A
30 IF A=0 THEN PRINT B/C
40 B=B+A
50 C=C+1
60 GOTO 20
```

Although the program input is completed, the program does not terminate after displaying the average. Therefore, add an END statement after the PRINT statement on line 30 by using a multistatement.

```
30 IF A=0 THEN PRINT B/C:END
```

Using an IF statement, a test is performed by the conditional expression as shown above.

• IF statement applications

In the above example, program progress was determined by one test. However, if there are several tests and all conditions must be satisfied, what is the solution?

For example, an arbitrary numerical value is to be entered and numerical values from 1 to 9 are to be selected. In other words, since the selected numerical values must be larger than 0 and smaller than 10, two conditions ("0 < variable" and "variable < 10") are required. This can be written on one line as follows.

```
IF 0 < variable THEN IF variable < 10 THEN .....
```

Although the same kind of statement can be used when there are three conditions or more, it is recommended that a maximum of two conditions be used since using more than two is too complicated and the line becomes too long.

NOTE

MULTISTATEMENT (:)

A multistatement is convenient when short assignment expressions are arranged on one line, or when there are several commands after THEN in an IF statement.

Example 1)

```
10 A=1
20 B=2
30 C=3
    } ⇒ 10 A=1:B=2:C=3
    :
```

Example 2)

```
50 C=A+B
60 D=A-B
70 E=A*B
80 F=A/B
    } ⇒ 50 C=A+B:D=A-B:E=A*B:F=A/B
    :
```

When a multistatement is used after THEN in an IF statement, it is executed only when a condition expression is true. Therefore, precautions shall be taken.

Example 3)

```
IF A<0 THEN A=10:B=20:.....
                Executed only when
                condition is true.
```

A multistatement is convenient when arranging a program. However, since one line is not easy to see if it is too long, make one line with an appropriate length, and write the remaining items on the next line.

3-3-3 Repeating A Program (FOR-NEXT statement)

If combined GOTO and IF statements are repeated a certain number of times, the program becomes too long and too complicated. When the number of repetitions is known, the program can be arranged in a more simple way. The command that performs this repetition is the FOR-NEXT statement. In the FOR-NEXT statement, the calculation between the FOR statement and the NEXT statement is repeated a specified number of times. The format of a FOR statement is as follows.

FOR control variable = initial value TO final value STEP increment

The format of a NEXT statement is as follows.

NEXT control variable

A numerical variable with only one character, such as A or B, can be used as a control variable in a FOR statement, but an array variable cannot be used. The initial value, final value, and the increment can be a numerical expression or numerical variable. The control variable is repeatedly changed, by the increment, from an initial value to a final value. The increment can be omitted and becomes 1 when omitted.

Store and execute the following program to easily understand the function of a FOR-NEXT statement.

```
10 INPUT A
20 FOR B=1 TO 10 STEP A
30 PRINT B
40 NEXT B
50 GOTO 10
```

This program is executed as follows.

Operation

```
RUN [EXE]
3 [EXE]
[EXE]
[EXE]
[EXE]
[EXE]
0.8 [EXE]
[EXE]
[EXE]
[EXE]
:
```

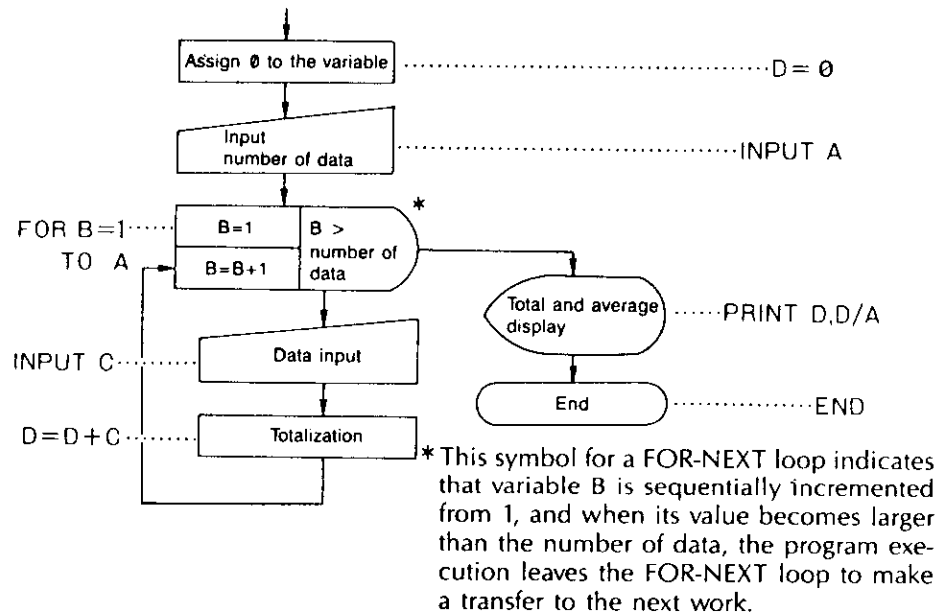
Display

?
1
4
7
10
?
1
1.8
2.6
3.4
⋮

Example) Prepare a program to obtain a total and an average for data when a certain number of data is entered.

For this program, enter the number of data first, then enter each data by a FOR-NEXT statement and obtain the total. After the data input has been terminated, the total and average are displayed.

Prepare the flowchart.



Prepare a program based on this flowchart.

```

10 D=0
20 INPUT A
30 FOR B=1 TO A
40 INPUT C
50 D=D+C
60 NEXT B
70 PRINT D,D/A
80 END

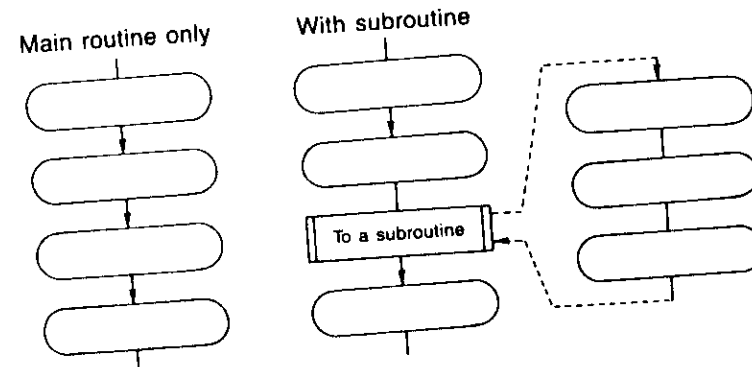
```

When the number of data is known as shown above, data input and calculation can be repeated by a FOR-NEXT statement. The number of data can be directly written instead of A on line 30 without entering the number of data by an INPUT statement as shown on line 20. In this case, line 20 is not required.

3-3-4 A Convenient Subroutine For Complicated Programs (GOSUB-RETURN statement)

You are now accustomed to preparing programs; they can become long and complicated. A program, which is convenient for performing repetitive processing and is especially helpful when composing lengthy programs, is called a "subroutine".

A subroutine recalled by a main routine performs part of the work.



Check its function by using an example.

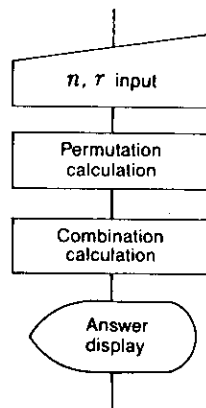
Example) Prepare a program to obtain permutations and combinations.

Expression: Permutations $P_r = \frac{n!}{(n-r)!}$

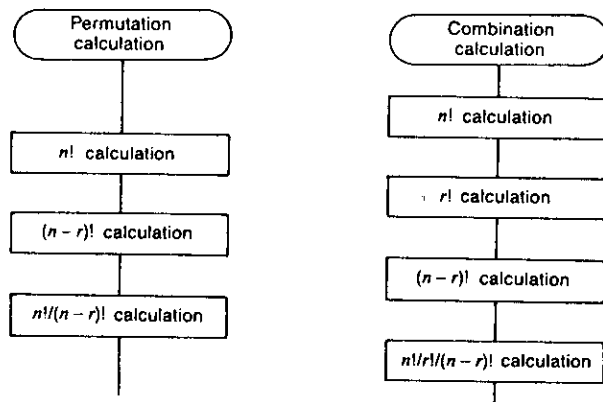
Combinations $C_r = \frac{n!}{r!(n-r)!}$

This program obtains permutations and combinations of two entered data items (n, r).

Prepare a simple flowchart.



Now prepare detailed flowcharts for the calculation of permutations and combinations.



The calculations of $n!$ and $(n - r)!$ are used in both of permutation and combination calculations. Also, three different factorial calculations are performed.

PROGRAM EXAMPLE (1)

```

10 INPUT N,R
20 A=1
30 FOR B=1 TO N
40 A=A*B
50 NEXT B
60 E=A
70 A=1
80 FOR B=1 TO N-R
90 A=A*B
100 F=A
110 NEXT B
120 P=E/F
130 A=1
140 FOR B=1 TO R
150 A=A*B
160 NEXT B
170 G=A
180 C=E/G/F
190 PRINT P,C
200 END
  
```

$n!$ calculation

$(n - r)!$ calculation

$\frac{n!}{(n - r)!}$ calculation

$r!$ calculation

$\frac{n!}{r!(n - r)!}$ calculation

VARIABLE CONTENTS

$N : n$
 $R : r$
 $P : \text{Permutation}$
 $C : \text{Combination}$
 $A : \text{For factorial}$
 $B : \text{For FOR-NEXT loop}$
 $E : n!$
 $F : (n - r)!$
 $G : r!$

In this program, three factorial calculations utilize a common program except for the final value in the FOR statements. If this final value can be controlled by a common variable, these three calculations can be made in common. Here using a subroutine is very effective. Use the variable H for the final values in order to make these calculations common. It is necessary to sequentially enter the values of n , $n - r$, and r into variable H.

Now this program is changed and includes a subroutine system; a command that transfers the work to the subroutine, and a command to provide a return after work termination are necessary. These commands are "GOSUB" and "RETURN".

PROGRAM EXAMPLE (2)

```

10 INPUT N,R
20 H=N
30 GOSUB 150
40 E=A
50 H=N-R
60 GOSUB 150
70 F=A
80 P=E/F
90 H=R
100 GOSUB 150
110 G=A
120 C=E/G/F
130 PRINT P,C
140 END
150 A=1
160 FOR B=1 TO H
170 A=A*B
180 NEXT B
190 RETURN

```

} Subroutine

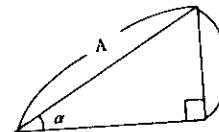
A program that has a common part can be simply prepared by using a subroutine as shown above.

Although this program is shorter by only one line, it has an important function when a program is more complicated or longer.

3-3-5 Using Functions

Functions can be used in a program. Some program examples are explained below.

Example 1) Trigonometric function



Obtain the length of side B of the triangle on the left by entering side A and angle α .

Expression: $B = A \cdot \sin \alpha$
Angle unit: Degree

PROGRAM EXAMPLE Operation example

```

10 MODE 4
20 INPUT A,D
30 B=A*SIN D
40 PRINT B
50 END

```

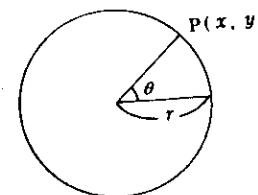
RUN **EXE**
(Side A) 15 **EXE**
(Angle α) 30 **EXE**

Display

?
?
7.5

The angle unit is specified on line 10. Since the calculation is performed in degree, "MODE 4" is specified. The trigonometric function is used to perform calculation on line 30.

Example 2) Trigonometric function



Obtain the coordinates (x, y) of point P of the circle on the left by entering the radius r and angle θ .

Expressions: $x = r \cdot \cos \theta$

$y = r \cdot \sin \theta$

Angle unit: Radian

PROGRAM EXAMPLE **Operation example**

```

10 MODE 5
20 INPUT R,T
30 X=R*COS T
40 Y=R*SIN T
50 PRINT X,Y
60 END

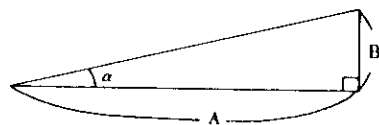
```

(Radius) 5
(Angle θ) $\pi/3$

Display

?
?
2.5
4.330127019

Since the angle unit is radian, "MODE 5" is specified on line 10. The x-coordinate and y-coordinate are obtained on lines 30 and 40.

Example 3) Inverse trigonometric function

Obtain angle α of the triangle on the left by entering side A and side B.

Expression: $\alpha = \tan^{-1} \frac{B}{A}$
Angle unit: Degree

PROGRAM EXAMPLE **Operation example**

```

10 MODE 4
20 INPUT A,B
30 D=ATN(B/A)
40 PRINT D
50 END

```

(Side A) 100
20

Display

?
?
11.30993247

Example 4) Decimal $\leftrightarrow$ Sexagesimal conversion

Prepare a program that performs time calculation.

PROGRAM EXAMPLE **Operation example**

```

10 T=0
20 INPUT D,E,G
30 S=SGN D
40 D=ABS D
50 T=T+S*DEG(D,E,G)
60 PRINT DMS$(T)
70 GOTO 20

```

(Hour) 1
(Minute) 25
(Second) 36
2
(Hour) 2
(Minute) 15
(Second) 5

Display

?
?
?
1°25'36
?
?
?
3°40'41

On line 20, enter the hour, minute, and second into 3 variables, D, E, and G respectively.

Lines 30 and 40 are used for subtraction. If the hour is entered with a negative sign (-), the entered time is subtracted from the previous result. If only addition is performed, these lines are not necessary.

The total is obtained on line 50. 1 is entered into variable S to perform addition, and -1 to perform subtraction. The DEG function converts sexagesimal (hours, minutes and seconds) to decimal, and totalization is performed in decimal.

Line 60 for the display uses the DMS\$ function that converts decimal to sexagesimal.

Example 5) Random number generation
Generate random numbers from 1 to 99.**PROGRAM EXAMPLE**

```

10 R=INT(RAN#*99)+1
20 PRINT R
30 GOTO 10

```

A random number is generated on line 10. The number being from 1 to 99 in this example, multiply the generated random number by 99 and add 1 to the result to obtain a number from 1 to 99 (see page 178).

When the random number is from 5 to 9 $\rightarrow$ INT(RAN#*5)+5

When the random number is from 10 to 20 $\rightarrow$ INT(RAN#*11)+10

3-3-6 Using An Array

Being different from general variables from A to Z, an array variable is managed by a number (subscript).

An alphabetical character from A to Z can be used for the variable name with a subscript attached.

Example)

A(1)

Subscript

Variable name

An array is convenient when a large amount of data is handled.
For example, to enter 10 data items:

```
10 FOR A=1 TO 10
20 INPUT N(A)
30 NEXT A
```

In this case, the array is arranged as follows.

General variables	O	P	Q	R	S	T	U	V	W	X
Array variables	N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)

To select the largest data item from among 10:

```
100 A=0
110 FOR B=1 TO 10
120 IF N(B)>A THEN A=N(B)
130 NEXT B
140 PRINT A
```

When an array is used, precautions should be taken concerning its arrangement. Some array containers are commonly utilized as those for general variables. For example, the container for A(0) is the same as that for A, and A(1) is the same as B. Therefore if A(0) and A are used in the same program, data in the variable is changed. The common parts are as follows.

A	B	C	D		X	Y	Z
A(0)	A(1)	A(2)	A(3)		A(23)	A(24)	A(25)
	B(0)	B(1)	B(2)		B(22)	B(23)	B(24)
		⋮				⋮	
					X(0)	X(1)	X(2)
						Y(0)	Y(1)
							Z(0)

For example, perform the following operations.

Assign 7 to variable I.

I = 7 **EXE**

Confirm the content of variable I.

I **EXE**

Assign 10 to the 9th container of array variable A.

A(8) = 10 **EXE**

Confirm the content of variable I.

I **EXE**

The content of variable I has been changed as shown above. It is because the container for variable I is the same as that for array variable A(8). When an array is used in a program, keep variables for FOR-NEXT loop or assignment, and then determine the variable names of this array.

Example 1) When 10 array variables are used,
Array variables: A(0) — A(9)
Unused general variables: K — Z

Example 2) When 50 array variables and 15 general variables are used,
General variables: A — O
Array variables: P(0) — P(49) [Operate DEFM 39 **EXE** .]

Example 3) When variable A is used for FOR-NEXT loop, variable B for totalling and 50 array variables are used,

```
10 DEFM 26
```

```
20 B=0
```

```
30 FOR A=0 TO 49
```

```
40 INPUT C(A)
```

```
50 B=B+C(A)
```

```
60 NEXT A
```

..... Increase variables by 26 to obtain
52 variables.

..... Array variables C(0) to C(49) can be used.

When a large amount of data is handled by an array, precautions shall be taken so that these array variables do not overlap other general variables.

Also, precautions shall be taken on memory expansion when an array is used. When the general variables A to Z are sufficient for the size of the array, expansion is not required. However, when these variables are not sufficient, be sure to expand the memory.

Memory expansion is performed by the DEFM command which specifies the amount of expansion.

For example, when an expansion of 10 variables is performed,

Manually: DEFM 10 **EXE**

In a program: Line number DEFM 10

This command can be used manually or in a program. Write it at the beginning of a program when an array is used. Now, analyze a program that uses an array.

Example) Assign random numbers from 0 to 59 to array variables from G(0) to G(59) and arrange them in the descending order for display.

PROGRAM EXAMPLE

```

10 DEFM 40
20 FOR B=0 TO 59
30 G(B)=INT(RAN#*60)
40 NEXT B
50 BEEP 1
60 FOR B=0 TO 59
70 A=-1
80 FOR C=B TO 59
90 IF G(C)>A THEN A=G(C):D=C
100 NEXT C
110 E=G(B):G(B)=G(D):G(D)=E
120 NEXT B
130 BEEP 0
140 FOR B=0 TO 59
150 PRINT G(B)
160 NEXT B
170 END

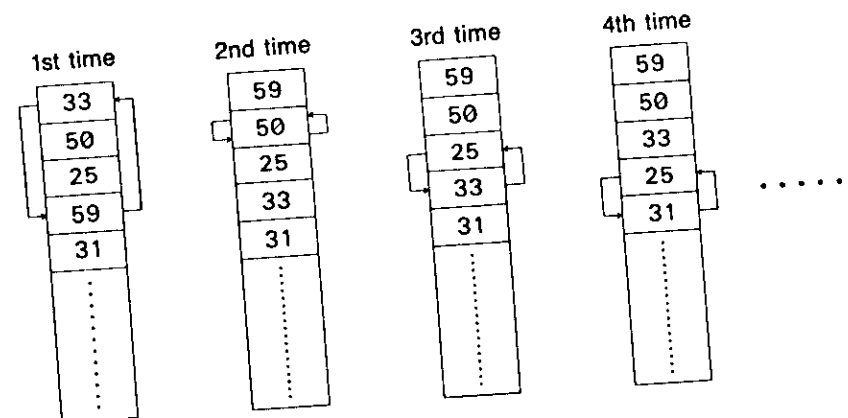
```

} Assign random numbers
0—59 to G(0)—G(59).

} Arrange them in the
descending order.

} Display them sequentially.

This program consists of three parts. In the first part, numerical values from 0 to 59 are generated as random numbers which are assigned to array variables G(0) to G(59). In the second part, they are arranged in the descending order. In the third part, they are displayed in this order. The BEEP statements on lines 50 and 130 are used to generate a buzzer sound. A high sound is generated by "BEEP 1" and a low sound generated by "BEEP 0". Since these two lines are only used to generate sound that indicates the termination of data preparation and arrangement, they can be deleted from the program. Lines 60 to 120 are repeatedly executed to arrange data in the descending order.



The DEFM statement on line 10 is used to expand the memory. Since 20 variables (G to Z) are remaining and 60 variables (G(0) to G(59)) are necessary, memory expansion for 40 of them is required. An array is convenient when a large amount of data is handled.

3-3-7 Data Read-in (READ, DATA, RESTORE statements)

When an INPUT statement (data input method) is used, "?" is displayed during program execution to ask data input from the keyboard. A READ statement reads data written in a DATA statement and assigns them to variables.

Example) Read 10 data items in a DATA statement and display them.

PROGRAM	Operation	
10 FOR B=1 TO 10	RUN EXE	1
20 READ A	EXE	2
30 PRINT A	EXE	3
40 NEXT B	...	...
50 DATA 1,2,3,4,5, 6,7,8,9,10	...	...
60 END	EXE	10

A READ statement must be used together with a DATA statement since a data item fetched from a DATA statement is assigned to a variable following "READ".

Several variables can be written after a READ statement by punctuating them with commas (,).

Example)

PROGRAM	Operation	Display
10 READ A\$,B\$	RUN EXE	CASIO PB & FX
20 PRINT A\$;B\$		
30 END		
40 DATA CASIO,PB & FX		

A DATA statement can be placed anywhere in a program. After program execution, data are assigned to variables sequentially from the first data of the DATA statement that has the smallest line number. When data written in a DATA statement are numerical values, use numerical variables for the variables in the READ statement, and when they are characters, use character variables.

Example)

PROGRAM	Operation	Display
10 RESTORE	RUN EXE	ABC
20 READ A\$	EXE	123456
30 PRINT A\$	EXE	ABC
40 READ B		
50 PRINT B		
60 RESTORE 90		
70 READ C\$		
80 PRINT C\$		
90 DATA ABC		
100 DATA 123456		
110 END		

When no line number is specified by a RESTORE statement, the first data in the first DATA statement in the program is read by the next READ statement. When a line number is specified, the first data in the specified DATA statement is read by the next READ statement.

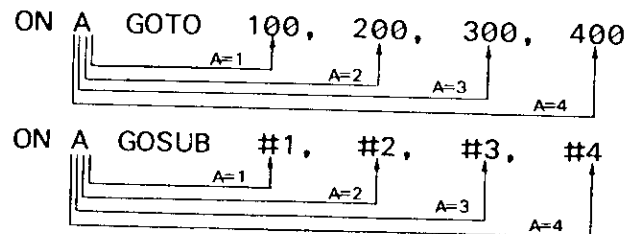
3-3-8 Indirect Specification (ON-GOTO, ON-GOSUB statements)

Although GOTO statement and GOSUB statement specifications are performed by writing the line number or program area directly in a program, sometimes the branching depends on the arithmetic result or the data for processing convenience. In this case, testing the condition with an IF statement is not convenient.

The commands that determine and specify a branching in a program are indirect specifications (ON-GOTO and ON-GOSUB).

The function of an ON-GOTO statement is similar to that of an ON-GOSUB statement.

Example)



Branching is performed to the 1st location if the value of A is 1, and to the 2nd location if this value is 2, etc. It depends on the numerical variable or the result of the calculation expression that follows "ON". When the number of branching locations is less than the value of the variable or calculation expression, or when a branching location does not exist, program execution advances to the next line, or command in case of a multistatement.

Example)

```

10 INPUT A
20 ON A GOTO 100,200,300,400,500
30 PRINT "NO"
40 END
100 PRINT "LINE 100":END
200 PRINT "LINE 200":END
300 PRINT "LINE 300":END
400 PRINT "LINE 400":END
500 PRINT "LINE 500":END

```

3-3-9 Character Handling Functions (LEN, MID\$, VAL, STR\$)

While SIN and COS are called numerical functions because they handle numerical values, there are character functions that handle characters. This computer is provided with "LEN", "MID\$", "VAL" and "STR\$" character functions.

• LEN

The LEN function counts the number of characters in a character variable.

Example)

Display

3

* An array variable cannot be used with the LEN function.

• MID\$

The MID\$ function fetches characters from among those stored in the exclusive character variable (\$) by specifying the starting location and the number of characters to be fetched.

Example)

```

10 $="CASIO PB&FX"
20 PRINT $
30 PRINT MID$(1,5)
40 PRINT MID$(7,5)
50 END

```

Operation

RUN

Display

CASIO PB&FX
CASIO
PB&FX

• VAL

The VAL function converts numerals stored in a character variable to a numerical value.

Example)

```

10 A$="123":B$="456"
20 PRINT A$+B$
30 PRINT VAL(A$)
  +VAL(B$)
40 END

```

Operation

RUN

Display

123456
579

* An array variable cannot be used with the VAL function.

• STR\$

The STR\$ function converts numerical values stored in a numerical variable to a character string; this is the inverse of the VAL function.

Example)

```
10 A=123:B=456
20 PRINT A+B
30 PRINT STR$(A)+STR$(B)
40 END
```

Operation

RUN 


Display

579
123456

Example)

```
10 INPUT $
20 FOR A=1 TO LEN($ )
30 PRINT MID$(A,1);
40 BEEP 1
50 NEXT A
60 END
```

This program fetches a character entered in the exclusive character variable (\$) with the MID\$ function. One character is fetched each time. The starting location is specified by a FOR-NEXT statement, and the final value is determined by the LEN function.

" ; " is added at the end of line 30 so that the program does not stop because a continuous display is desired.

Example)

```
10 A=1:B=0
20 PRINT "<";STR$(A);">";
30 INPUT $
40 IF $="END" THEN 100
50 B=B+VAL($ )
60 A=A+1
70 GOTO 20
100 PRINT B/(A-1)
110 END
```

This program obtains the average for an unknown number of data. Data input is terminated by entering END, and the average is displayed by branching to line 100.

Line 20 provides a message that enables easier input.

On line 50, since data is entered into the exclusive character variable (\$) as a character, totalization is performed after converting it to a numerical value. Also, since data input is terminated by entering END, an error (ERR2) occurs if anything else is entered.

3-3-10 Input/Output Control Functions (KEY\$, CSR)

Although the KEY\$ function is used to enter data, it differs from an INPUT statement as follows.

INPUT statement	KEY\$ function
• Within a 12 digit mantissa and a 2 digit exponent for a numerical value. • Up to 7 characters for a character variable and up to 30 characters for the exclusive character variable (\$).	• Reads the character of the key which has been pressed and assigns it to a character variable.
• Input waiting is indicated by a "?" display.	• No input waiting display occurs.

Example)

```
10 INPUT A
20 PRINT A
30 B$=KEY$
40 IF B$=" " THEN 30
50 PRINT B$
60 END
```

Line 10 uses an INPUT statement while lines 30 and 40 utilize the KEY\$ function. The KEY\$ function accepts input of one character from the keyboard, but no input waiting display occurs and the execution does not stop. Therefore, this function is combined with an IF statement as shown on line 40, and if character input is not performed, a return is made to line 30.

Example)

```

10 A$=KEY$
20 IF A$="1" THEN 100
30 IF A$="2" THEN 200
40 IF A$="3" THEN 300
50 GOTO 10
100 PRINT "LINE 100":END
200 PRINT "LINE 200":END
300 PRINT "LINE 300":END

```

In the previous example a check was made for key depression. In this program a check is made for the keyboard entry of 1, 2 or 3. If a condition is true, an advance is made to the next work.

When the KEY\$ function is used at the beginning of a program like in this example, pay attention to program starting. There are two different program starting methods. When  is used for a program starting method, unless the  key is released immediately, the numeral 1 is read by the KEY\$ function; "LINE 100" will be displayed.

When the KEY\$ function is used at the beginning of a program, add the following lines.

```

5 A$=KEY$
6 IF A$=" " THEN 5
10 A$=KEY$
  :
  :
  :

```

} Waits until the pressed key is released.

The CSR function specifies a data display location; it is used in a PRINT statement.

Example)

```

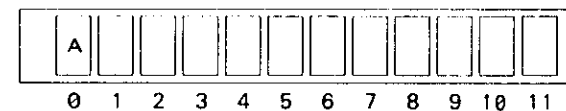
10 PRINT "A"
20 PRINT CSR 2;"A"
30 PRINT CSR 8;"A"
40 END

```

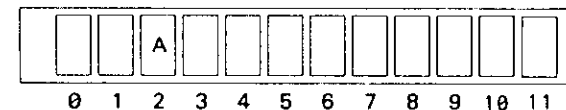
The CSR function can be understood by executing this program.

When this function is used, display starts from the specified location (0, 1, 2, ... or 11 from the left).

When it is not used, display starts from the extreme left.



PRINT "A"



PRINT CSR 2;"A"



PRINT CSR 8;"A"

Example)

```

10 A=INT(RAN#*12)
20 PRINT CSR A;"1"
30 GOTO 10

```

This program generates a numerical value from 0 to 11 by using the RAN # function and displays "1" with the CSR function. After a certain time, "1" is displayed at different locations. An interesting game can be prepared by combining the KEY\$ and CSR functions.

3-4. CONVENIENT OPTIONAL EQUIPMENTS

3-4. CONVENIENT OPTIONAL EQUIPMENTS

This computer has optional equipments: a cassette tape recorder interface (FA-3, FA-5), interface pack for connecting the FP-40 (SB-42) and a character mini printer (FP-12S).

The FA-3/FA-5 allows programs and data to be stored from the computer on a tape, or to be loaded from a tape to the computer.

The FP-12S/FP-40 can print programs, data and arithmetic results.

The functions of these equipments are explained below.

3-4-1 Storing A Program Or Data

To store a program or data on cassette tape, the FA-3 or FA-5 is necessary. To connect the FA-3 or FA-5 to the computer and to a tape recorder, see the Instruction Manual that comes with the FA-3 or FA-5.

■ Program storing and loading

Since programs are memorized in the computer, sometimes the next program cannot be memorized because of memory capacity. If the previous program is erased, it cannot be used again. In this case, the FA-3/FA-5 is very helpful.

Commands for storing programs on a tape are "SAVE" or "SAVE ALL". "SAVE" can only store a program located in one program area, while "SAVE ALL" can simultaneously store programs located in all program areas.

SAVE command



READY P_n

→ The program located in this program area can be stored.

SAVE ALL command

Programs located in all program areas can be stored.

The SAVE and SAVE ALL commands are manually executed.

Example)

```
SAVE 
SAVE "CASIO" 
SAVE ALL 
SAVE ALL "PB" 
```

Characters enclosed with the quotation marks (") after SAVE and SAVE ALL are file names which are placed with stored programs. These programs can be loaded later by specifying these names. Up to 8 characters can be used for a file name.

LOAD and LOAD ALL commands are used to load programs from a tape to the computer. The proper use of these commands depends on whether programs were stored by SAVE or SAVE ALL.

Loading Storing	LOAD	LOAD "file name"	LOAD ALL	LOAD ALL "file name"
SAVE	○	×	×	×
SAVE "file name"	○	○	×	×
SAVE ALL	×	×	○	×
SAVE ALL "file name"	×	×	○	○

* Items marked with "○" can be loaded; those marked with "X" cannot be loaded.

* File names must be identical.

Example)

```
LOAD 
LOAD "file name" 
LOAD ALL 
LOAD ALL "file name" 
```

When programs are loaded by LOAD or LOAD ALL, a display depending on the storing format appears.

Storing format	Display
SAVE	PF:
SAVE "file name"	PF: file name
SAVE ALL	AF:
SAVE ALL "file name"	AF: file name

A program stored by a SAVE command can be loaded to any of the program areas by a LOAD command.

Example) Stores the program of P0.

↓
Loads it to P9.

Precautions:

Sometimes a program cannot be stored or loaded smoothly. If this happens, check the following items.

- "ERR9" is displayed during storing.
[Check point]
Check if the computer is properly connected to the FA-3 or FA-5.
- "ERR9" is displayed during loading.
[Check points]
If the tape is stretched, replace it with a new one.
If the head of the tape recorder is dirty, clean it.
Set the tone control of the tape recorder to medium.
- No error is displayed but loading is attempted without success.
[Check points]
If the tape recorder output volume is low, increase the volume near MAX.
Check if the output standard of the tape recorder is in accordance with that of the FA-3 or FA-5.

■ Storing and loading of data for DATA BANK function

When stored data for DATA BANK function are transferred to another unit, they must be stored on tape.

"SAVE #" is used to store all the data for DATA BANK function at once.

SAVE # "file name"

Up to 8 characters.

Up to 8 characters can be placed inside " " for a file name, the same as when a program is stored.

Example)

SAVE # "MEMO" **EXE**

To load data for DATA BANK function from tape to the computer, "LOAD #" is used.

The previous data are erased when new data are stored.

LOAD# "file name"

Up to 8 characters.

Example)

LOAD# "MEMO" **EXE**

When data for DATA BANK function are being loaded, a display depending on the storing format appears.

Storing format	Display
SAVE #	MF:
SAVE # "file name"	MF: file name

■ Data storing and loading

A program always has data; it is troublesome to enter these data from the keyboard each time.

Try a method by which data in the memory are stored on tape and loaded again.

To store data on a tape, "PUT" is used.

Variables are specified in a PUT command. A file name can also be specified.

PUT "file name" \$, A, Z

Up to 8 characters. Stores 26 variables from A to Z.

For a file name, up to 8 characters can be placed inside " " as for program storing.

If the exclusive character variable (\$) is used, specify it first. Then next two variable names are specified to determine the beginning and end of the variables to be stored.

Example)

Store the content of the exclusive character variable (\$) and 13 variables from A to M.

PUT \$, A, M

Store the content of 36 variables from A to Z(10) with a file name, "CASIO".

PUT "CASIO" A, Z(10)

Since the variable names specify the beginning and end of the variables to be stored, place them in alphabetical order (e.g., "A, Z"). A specification such as "Z, A" cannot be performed.

When the variables are character variables, "A, Z" can be specified instead of "A\$, Z\$".

"GET" is used to load data from a tape to the computer. Variables are specified in a GET command. A file name can also be specified.

GET "file name" \$, M, W

Up to 8 characters. Loads to variables from M to W.

Example)

Load data to the exclusive character variable (\$) and 3 variables from X to Z.

GET \$, X, Z

Load data with a "PB" file name to variables from G(0) to G(59).

GET "PB" G(0), G(59)

When data is being loaded by a GET command, a display depending on the storing format appears.

Storing format	Display
PUT \$, A, Z	DF:
PUT "file name" G, P	DF: file name

3-4-2 Keeping A Record

If the content of a program can be printed after preparation, it is convenient to perform debugging or to modify its content. It is also convenient to be able to keep an arithmetic result after it is printed.

Perform printing with the FP-12S character mini-printer or FP-40 character printer with SB-42 interface pack.

To perform printing the print mode ("PRT" is displayed) must be specified by pressing **MODE** . This mode can be released by pressing **MODE** .

To print the content of a program, execute the LIST command after pressing **MODE** .

Example)

Input the following program.

```
10 INPUT A
20 PRINT A*A
30 GOTO 10
```

Then perform printing.

MODE 7 LIST EXE

Print example

```
LIST
10 INPUT A
20 PRINT A*A
30 GOTO 10
```

When the contents of all program areas are to be printed, perform

LIST ALL **EXE**

Release the print mode by pressing **MODE**  after printing is completed. To print an arithmetic result, specify the print mode by pressing **MODE**  or by writing "MODE 7" in the program. When **MODE**  is pressed, all key operations after that are printed. Therefore, if only one part is to be printed, it is more convenient to write "MODE 7" in the program.

* In this case, input must be performed by pressing the **M O D E** keys instead of the **MODE** key.

Example)

Only the arithmetic result is printed.

```
10 INPUT A
20 MODE 7
30 PRINT A*A
40 MODE 8
50 GOTO 10
```

In this program, the print mode is specified after data input and is released after printing to return to input waiting status.

The display by the PRINT statement is not stopped during printing, and the program execution advances to the next command after printing. Release the print mode by writing "MODE 8".

3-5. USING A PB-100 PROGRAM

Programs prepared for the PB-100 and PB-300 can be utilized with this computer.

This computer is provided with more commands than the PB-100/PB-300; its utilization is more convenient.

The BASIC language used by this computer is almost the same as that used by the PB-100/PB-300.

- Different points

- **Additional commands**

- PASS (Program protection)
- BEEP (Buzzer sound)
- READ (Reads data from a DATA statement)
- DATA (Writes data)
- RESTORE (Specifies data to be read)
- ON-GOTO (Indirect specification of a GOTO statement)
- ON-GOSUB (Indirect specification of a GOSUB statement)
- REM (Comment statement)

- **Additional functions**

- DEG (Sexagesimal → decimal conversion)
- DMS\$ (Decimal → sexagesimal conversion)
- STR\$ (Converts a numerical value to a character string)

- **Modified commands**

This computer	PB-100/PB-300
NEW (NEW ALL)	CLEAR (CLEAR A)
CLEAR	VAC
IF—THEN	IF—;
SAVE ALL	SAVE A
LOAD ALL	LOAD A
VERIFY	VER
DEFM (Can be written in a program)	DEFM (Can only be performed manually.)

• Modified functions

This computer	PB-100/PB-300
KEY\$ MID\$	KEY MID

In spite of these different points, a program prepared by the PB-100/PB-300 can be fundamentally utilized with this computer.

However, it is better that programs be rewritten for this computer so that it can be easily used or can be easily reconsidered later.

Example 1)

PB-100 program

```

10 VAC
20 FOR A=1 TO 20
30 INPUT Z(A)
40 IF Z(A)>80;B=B+1:GOTO 90
50 IF Z(A)<60;C=C+1:GOTO 90
60 IF Z(A)>40;D=D+1:GOTO 90
70 IF Z(A)>20;E=E+1:GOTO 90
80 F=F+1
90 NEXT A

```

This example is part of a program to enter data and distribute them according to their size. Although the program could be used as it is, correct the following items.

Change "VAC" on line 10 to "CLEAR".

```

10 CLEAR

```

Change ";" on lines 40 to 70 to "THEN".

```

40 IF Z(A)>80 THEN B=B+1:GOTO 90

```

Since memory expansion is necessary in this program, write the DEFM command, manually executed in the PB-100/PB-300, at the beginning.

```

5 DEFM 20

```

Example 2)

PB-100 program

```

10 INPUT "I=1/O=2/P=3",N
20 IF N<1 THEN 10
30 IF N>3 THEN 10
40 GOTO N*100

```

This program is used to determine branch locations according to the work. To adapt it for this computer, modify it as follows by using an ON-GOTO statement.

```

10 INPUT "I=1/O=2/P=3",N
20 ON N GOTO 100,200,300
30 GOTO 10

```

The program is simplified by utilizing an ON-GOTO statement as mentioned above; testing the data N is deleted. Programs and data stored on tape by CASIO'S handheld computers can be loaded as they are to this computer. However, the reverse operation is not always possible. Therefore precautions shall be taken. The relationships are as follows.

This computer → PB-100F, FX-710P

LOAD \ SAVE	PF	AF	MF	with password		
				PF	AF	MF
LOAD	○	○	○	○	○	○
LOAD ALL	○	○	○	○	○	○

This computer → PB-100, PB-300, FX-700P, FX-802P

LOAD \ SAVE	PF	AF	MF	with password		
				PF	AF	MF
LOAD	○	○	○	○	○	○
LOAD ALL	○	○	○	○	○	○

This computer → PB-110, PB-220, PB-410, FX-720P, FX-820P, FX-730P, FX-770P, FX-785P, FX-790P, FX-795P

LOAD \ SAVE	PF	AF	MF	with password		
				PF	AF	MF
LOAD	○	⧘	⧘	○	⧘	⧘
LOAD ALL	⧘	○	⧘	⧘	○	⧘
LOAD #	⧘	⧘	○	⧘	⧘	○

○ : Can be loaded.

⧘ : Cannot be loaded.

[IMPORTANT]

- When a program prepared by this computer is transferred to other CASIO's computer (excluding PB-110, PB-220, PB-410, FX-720P, FX-730P, FX-770P, FX-785P, FX-790P, FX-795P, FX-820P), READ #, WRITE # and RESTORE # commands must not be used. KEY\$ and MID\$ should be changed to KEY and MID for the PB-100, PB-300, FX-700P, and FX-802P.
- When a program prepared by other CASIO's computers is executed with this computer, sometimes it cannot be properly executed as shown below.
 - * If a numerical expression is used for a branch location in an IF-THEN statement, an error occurs. In this case, change it to an IF-THEN-GOTO statement.
- When using PB-100 programs
 - * An exclusive character variable is stored on cassette tape using the PUT command with a PB-100 Series (excluding PB-220, FX-730P, FX-785P, FX-790P, FX-795P) unit, can be loaded by the PB-120, as long as the string contained by the exclusive character variable is not longer than 30 characters. PB-100 series units, however, cannot read exclusive character variables stored by the PB-120.
 - * When transferring a program prepared with other CASIO's computers to this computer, a DIM or ERASE command must not exist in the program.

CHAPTER 4

DATA BANK FUNCTION

CASIO's handheld computer with the DATA BANK function allows you to use it without programming. You can easily enter data from the keyboard. The computer processes data speedily and accurately. From now on, you have access to an accurate and convenient storage unit. This chapter summarizes the usage of the data bank function to help you build your own DATA BANK.

4-1. HOW TO USE THE DATA BANK FUNCTION

We have explained the fundamental key operation in the previous chapter. However, unless you can activate the computer function, you cannot use it. Learning about the fundamental operation is a warming-up exercise. Now, let's start a practical exercise.

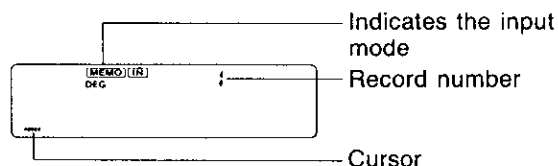
As explained before, the DATA BANK function permits you to activate the computer without programming as if you were using a calculator. You have only to enter data to the computer.

Now, let's enter data into the computer as a personal telephone book and learn how to use it.

4-1-1 Specifying the Input Mode for DATA BANK Function

To use the DATA BANK function, you must enter data. Before entering data, you must specify the input mode for DATA BANK function.

Press the **MODE** key, then the numerical key **3**. The mode will change to the input mode as shown below.



MEMO (IN) is displayed to indicate that the current mode is the input mode. The number "1" is called a record number. The cursor blinks at the left end to indicate that the computer is waiting for you to enter data. When the record number is 1, no data is stored; when it is 5, four records have already been stored.

4-1. HOW TO USE THE DATA BANK FUNCTION

4-1-2 Entering Data

Now, let's enter ten person's names and their telephone numbers.

ANDERSON	312-780-1234	JOHNSON	214-384-3569
BUSCH	800-562-7386	JONES	713-461-7132
COOK	414-785-6602	COLLINS	314-837-1007
GATES	612-536-4321	SMITH	615-259-2935
JACKSON	514-364-0038	WILSON	403-352-6221

Press the alphabetic keys **A N D E R S O N**, followed by **3**, then **1 2 3 4 5 6 7 8 9 0 1 2 3 4**.

Since the display window is 12 characters wide, the characters are shifted to the left, one character at a time, when the total number of characters exceeds 12. When the first record has been entered, the display will appear as shown:

OPERATION	DISPLAY
A N D E R S O N	MEMO (IN) DEG ANDERSON_
3	MEMO (IN) DEG ANDERSON, _
3 1 2	MEMO (IN) DEG DERSON, 312-_
7 8 0 - 1 2 3 4	MEMO (IN) DEG 12-780-1234_

If entered data is incorrect, move the cursor with the cursor control keys, $\leftarrow$ and $\rightarrow$, and enter the correct data again. Check that the data is correct and press the **EXE** key. Thus, the data is stored in the computer as record 1. The following display appears with the cursor erased.

EXE

(MEMO) (1R)
DEG 1
ANDERSON, 312

Pressing the **EXE** key again clears the display, changes the record number to 2, and blinks the cursor to wait for you to enter the next data. Next, enter BUSCH and his telephone number.

B U S C H
?

(MEMO) (1R)
DEG 2
BUSCH, _

8 0 0 -

(MEMO) (1R)
DEG 2
BUSCH, 800- _

5 6 2 -
7 3 8 6

(MEMO) (1R)
DEG 2
00-562-7386 _

EXE

(MEMO) (1R)
DEG 2
BUSCH, 800-56

• Input of $\rightarrow$ key

As shown in the above two examples, a name and a telephone number are separated by $\rightarrow$. The data in a record can be separated by the $\rightarrow$ key. Be sure to enter this key because it is very important when retrieving data.

Enter the names and telephone numbers of the remaining persons, from COOK to WILSON, in the same way as those of ANDERSON and BUSCH.

After entering all data, press the **EXE** key again to clear the display.

Note:

When entering data continuously, you don't have to clear the display by pressing the **EXE** key twice each time you enter data for one record. Pressing the **EXE** key once will cause the data to be stored. When the first character of the next data is entered, the previous data on the display disappears. However, until the **EXE** key is pressed, the previous record number remains displayed.

4-1-3 Searching Data

Simply press the **MEMO** key to retrieve the stored data.

1 Search operation with the **MEMO** key

Press the **MODE** key, then the numerical key **0**. The **RUN** mode is activated and the display will be as shown (Immediately after the power is turned on, the **RUN** mode is specified automatically).

RUN DEG
READY P0

This display means that the program area **P0** is available for use. The computer is waiting for an instruction to be entered.

First, press the **MEMO** key. The display will change as shown, and the name and part of the telephone number of ANDERSON (record 1) will appear.

MEMO
RUN (MEMO) DEG
1
ANDERSON, 312

Press the **MEMO** key again. Only the telephone number of ANDERSON will appear.

MEMO
RUN (MEMO) DEG
1
312-780-1234

Press the **MEMO** key again, the name and part of the telephone number of BUSCH (record 2) will appear.

MEMO
RUN (MEMO) DEG
2
BUSCH, 800-56

As shown above, the computer displays the first 12 characters of each data item in the same record which is separated by a comma. After all the data in that record are displayed, the computer processes the next record. Thus, the **MEMO** key is used to display comma-separated data in record number sequence.

After all the recorded data are displayed, the computer waits for the next data to be entered. If you press the **MEMO** key at this time, the computer displays data again, starting from record 1.

2 Quick search operation with the **EXE** key

To quickly search for data and display it, use the **EXE** key.

Pressing the **MEMO** key will display the data in record 1. Then, press the **EXE** key. The first 12 characters of the data in record 2 will appear. Press the **EXE** key again and the first 12 characters of the data in record 3 will appear (See the figure given below).

This operation can be repeated until the last record number and the corresponding data appear.

MEMO
RUN (MEMO) DEG
1
ANDERSON, 312

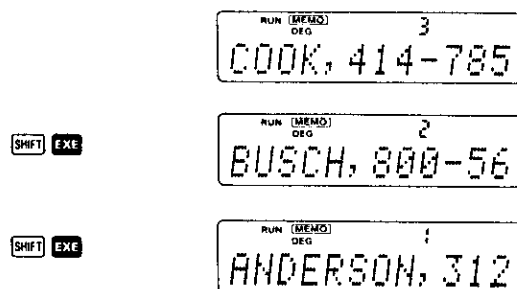
EXE
RUN (MEMO) DEG
2
BUSCH, 800-56

EXE
RUN (MEMO) DEG
3
COOK, 414-785

As explained above, the **MEMO** key displays all the data in a record and then the next record, while the **EXE** key displays only the first 12 characters of a record and then the next record.

3 Backward search operation with the **SHIFT** and **EXE** keys

Both the **MEMO** key and the **EXE** key search for data in the forward direction of record numbers, but not in the backward direction. To display the previous record during the search operation, press the **SHIFT** key and the **EXE** key. These keys display the first 12 characters of the data in the previous record. This operation can be repeated until record 1 is displayed.



4 Conditional search operation

When a small number of data are stored, it is possible to sequentially search the entire data from the beginning with the **[MEMO]** key or to display one record at a time with the **[EXE]** key. As the amount of data grows, however, it takes longer.

To speed up the operation, you can perform the conditional search operation which displays only the data you need.

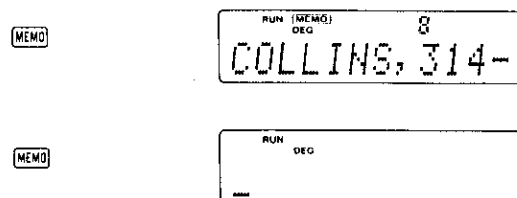
For example, you may wish to find the names beginning with C and their telephone numbers or to find WILSON's telephone number.

First, search for the names beginning with C.

Enter **[C]** and press the **[MEMO]** key. COOK's data (record 3) will appear.



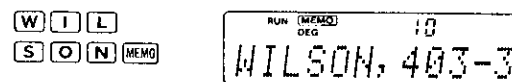
Pressing the **[MEMO]** key displays COLLINS' data (record 8).



Press this key again. The display is cleared and the cursor blinks to indicate that the computer is waiting for the next data to be entered. This means that the search for the names beginning with C has ended and that there is no such data any more. If there are multiple data items satisfying the specified condition, they are displayed in the ascending order of record numbers.

Let's find WILSON's telephone number in the same manner.

Enter **[W][I][L][S][O][N]** and press the **[MEMO]** key. WILSON's data will appear.



When performing the conditional search operation, you don't have to display data with the **[MEMO]** key or to clear the display in advance. Enter a search condition and press the **[MEMO]** key. Data satisfying the condition will appear immediately.

We have successfully found WILSON's data with the conditional search method, but we know only his toll number. In the case of the sequential search operation, pressing the **[MEMO]** key displays the next data item that follows the comma.

In the case of the conditional search operation, pressing the **[MEMO]** key will attempt to find WILSON's data. If WILSON's data exist, it appears; otherwise, the computer waits for the next data to be entered. This is the same as for the conditional search for the names beginning with C described above. Then, how can we display WILSON's telephone number?

5 Confirmation of data with the cursor control keys (**[←]** **[→]**)

When a data item is long, you can view the whole data item by shifting the displayed data to the right or left, one character at a time, with cursor control keys. Note that this is effective only within the same record. Shifting stops when the beginning or the end of the record is reached. When data is equal to or less than 12 characters, this function does not work.

RUN (MEMO) DEG 10
WILSON, 403-3



RUN (MEMO) DEG 10
403-352-6221

6 Confirmation of data with the **SHIFT** and **SHIFT** keys

Pressing the cursor control key in the shift mode will display the next item that follows the comma. This is functionally equivalent to pressing the **MEMO** key. The **MEMO** key displays the next record after all the data in the record are displayed, while the **SHIFT** and keys are effective only within that record.

The **SHIFT** and keys will display the previous item that precedes the comma. These keys are also effective only within that record.

RUN (MEMO) DEG 10
WILSON, 403-3



RUN (MEMO) DEG 10
403-352-6221

7 Additional conditional-search

Assume that COOK's data is displayed as a result of the search for the names beginning with C. You can add another condition at this moment. Look at the figure below. The contents of record 3 (COOK's data) are displayed. Now let's search records 4—10 for the persons whose telephone numbers begin with 61.



RUN (MEMO) DEG 3
COOK, 414-785

Enter the condition **61**, then press **SHIFT** and **MEMO**. The condition has changed from "C" to "61". The telephone numbers beginning with 61 are retrieved from records 4—10 and are displayed.



RUN (MEMO) DEG 4
612-536-4321

Only the telephone number satisfying the condition is displayed. To display the name, press **SHIFT** and . The name stored in record 4, **GATES**, will appear.



RUN (MEMO) DEG 4
GATES, 612-53

Pressing **MEMO** again will start searching for the next telephone number beginning with 61. SMITH's telephone number (record 9) is displayed.



RUN (MEMO) DEG 9
615-259-2935

Thus, the additional conditional-search operation, activated by the **SHIFT** and **MEMO** keys, allows you to find a record satisfying the initial condition and, after altering the condition, to retrieve data satisfying that condition from the subsequent data. This function enables you to narrow down the search by altering the condition.

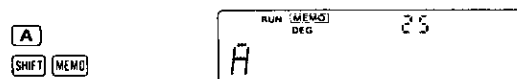
Assume that we have ranking data of test results.

Each data item consists of three fields: ranking, name, and class name. We will find the students in class A whose ranking are equal to or lower than 20. First, enter **20** and press the **MEMO** key to display the data of the student who ranks 20th.

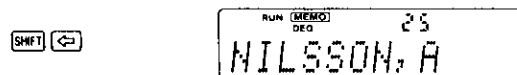


RUN (MEMO) DEG 20
20, MILLER, B

Enter **[A]** (class A), then press the **[SHIFT]** and **[MEMO]** keys. "A" indicating class A will appear.



You can check the name of the student, who ranks 20th or lower, in class A with the use of the **[SHIFT]** and **[<]** keys.



If the records are arranged in the order of ranking, a record number indicates its corresponding ranking.

Press the **[MEMO]** key to display the next data satisfying the condition in class A. When there is no more data satisfying the condition, the cursor blinks to indicate that the search has ended and that the computer is waiting for the next data to be entered.

Note:

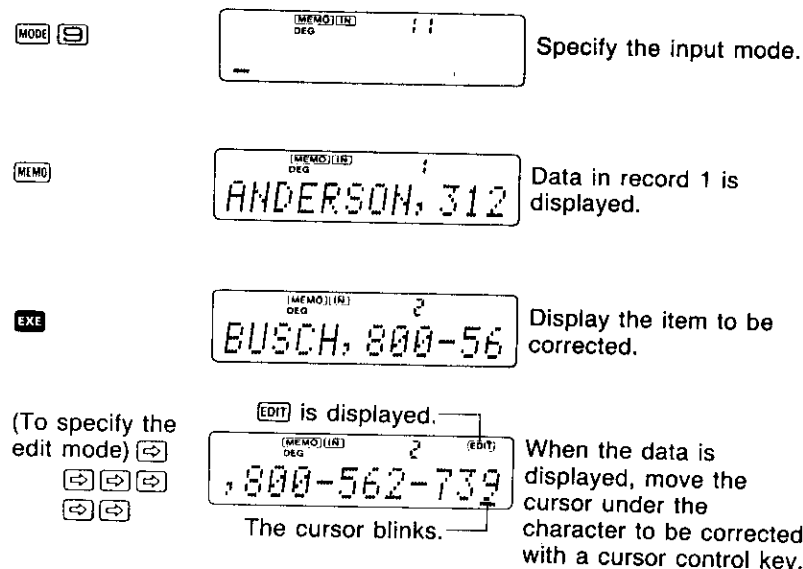
When you perform the conditional search operation, a specified search condition must agree with the stored data. For example, if data is stored in lowercase characters, the condition must be specified also in lowercase characters. Otherwise, the computer cannot find a match in the stored data. We can take advantages of this feature; i.e., the same data can be entered both in uppercase characters and in lowercase characters to assign it different meanings.

Failure to press the **[SHIFT]** key during the additional conditional-search operation (e.g., **[<] 1 [MEMO]**) will cause the computer to start searching for a data item satisfying a specified condition, assuming that a new condition has been specified. Be sure to press the **[SHIFT]** key.

4-1-4 Correcting Data

Data is corrected in the input mode for DATA BANK function. After entering data and before pressing the **[EXE]** key, move the cursor under the character to be corrected with a cursor control key, correct it, and press the **[EXE]** key.

You sometimes find incorrect data after the **[EXE]** key is pressed, for example, during the search operation. Assume that you entered 800-562-7396 instead of 800-562-7386 as BUSCH's telephone number. In this case, execute the following procedure:



When a cursor control key is pressed in the input mode, **EDIT** is displayed as shown, and the cursor and the character above it blink alternately. This indicates that the computer is in the edit mode. This mode activates the correction function and temporarily deactivates the search function.

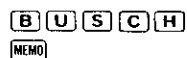


MEMO (IN) DEG 2
BUSCH, 800-56

Pressing the **EXE** key stores the correct data and displays the first 12 characters.

After data correction, **EDIT** disappears and the search function is activated again.

In the above example, the record to be corrected was displayed in the input mode with the **MEMO** and **EXE** keys. You can also display it by performing the conditional search operation as shown below. Conditional search is especially useful when much data is stored.



MEMO (IN) DEG 2
BUSCH, 800-56

Make sure that the data has been corrected.



MEMO (IN) DEG 2
800-562-7386

7396 has been corrected to 7386.

4-1-5 Deleting Data

To delete data, specify the input mode and press one of the cursor control keys, **←** and **→**, to specify the edit mode.

Either part of a record or all the contents of a record can be deleted.

1 Deleting part of a record

Display the data to be deleted and press a cursor control key to specify the edit mode.

Next, move the cursor to the character to be deleted.

Make sure that the cursor and the character blink alternately, press the **DEL** key, then press the **EXE** key.

The character will be deleted. The first 12 characters in the record appear and **EDIT** disappears.

To delete two or more characters, delete all unnecessary characters with the **DEL** key and press the **EXE** key.

2 Deleting all the contents of a record

To delete all the contents of a record, first specify the edit mode with a cursor control key.

Next, press the **AC** key and the **EXE** key.

All the contents of the record will be deleted, and the current record number will be automatically assigned to the next record.

• Deleting all stored data

As explained above, you can delete part of a record or all the contents of a record. You can also delete all the stored data.

First, specify the WRT mode by pressing **MODE** and **1**.



WRT mode
No. of remaining steps
The cursor blinks.

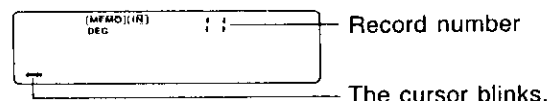
WRT appears indicating that the WRT (write) mode has been specified. **7457** indicates the remaining amount of memory capacity. **ERR1** appears when entering data causes the memory shortage. For the memory capacity, see the "NOTE—How to count the number of steps" on page 50. Enter **N** **E** **W** **SHIFT** **⏏** (**NEW #**), then press the **EXE** key. All the stored data will be deleted. Before deleting data, make sure it may be deleted.

4-1-6 Adding Data

You can add data next to the last record or insert a new record between two records.

1 Adding data

MODE 



When the input mode is specified, the computer waits for data to be entered and displays a record number. In the above figure, the computer displays the record number 11, blinks the cursor, and is waiting for data for record 11 to be entered. This means that ten records have already been stored. Enter new data and press the **EXE** key. Data will be entered into record 11.

2 Inserting data

The ten names, which were entered before, are not arranged in alphabetic order. To sort them into alphabetic order, you have to insert COLLINS' data between BUSCH and COOK and to delete record 8 which contains COLLINS' data.

First, specify the input mode and display the contents of record 3 (COOK's data) with the **MEMO** key. A new record can be inserted before the record being displayed.

MEMO (R) DEG 3
COOK, 414-785

Next, enter COLLINS, 314-837-1007.

C O L L
I N S
3 1 4
8 3 7
1 0 0 7

MEMO (R) DEG 3
14-837-1007

Pressing the **EXE** key at this moment would add the entered data next to the last record (in this case, record 11).

To insert a record between two records, press the **SHIFT** and **EXE** keys.

SHIFT EXE

MEMO (R) DEG 3
COLLINS, 314-

COLLINS' data has been entered into record 3, and each of the following records, beginning with COOK's data, is moved down one record. Press the **MEMO** key to make sure that COOK's data is displayed with record number 4.

Next, press the **EXE** key to display COLLINS' data. Since one record has been inserted, COLLINS' data is displayed with record number 9.

Press a cursor control key to specify the edit mode and press **AC EXE**. Record 9 will be deleted and sorting will be completed.

The following shows output examples before and after insertion.

Before insertion

```

LIST#
1 ANDERSON,312-78
  8-1234
2 BUSCH,800-562-7
  386
3 COOK,414-785-66
  82
4 GATES,612-536-4
  321
5 JACKSON,514-364
  -8838
6 JOHNSON,214-364
  -3569
7 JONES,713-461-7
  132
8 COLLINS,314-837
  -1887
9 SMITH,615-259-2
  935
10 WILSON,403-352-
  6221

```

After insertion

```

LIST#
1 ANDERSON,312-78
  8-1234
2 BUSCH,800-562-7
  386
3 COLLINS,314-837
  -1887
4 COOK,414-785-66
  82
5 GATES,612-536-4
  321
6 JACKSON,514-364
  -8838
7 JOHNSON,214-364
  -3569
8 JONES,713-461-7
  132
9 SMITH,615-259-2
  935
10 WILSON,403-352-
  6221

```

4-2. HOW TO USE THE DATA BANK FUNCTION IN A PROGRAM

To run a computer, a program containing instructions is necessary. The DATA BANK function of this computer is very useful as described before; it will become more convenient by combining BASIC program.

A program enables you to sort data in alphabetic order, to display data selectively, or to calculate data according to a specific rule and display the result and so on.

We should prepare programs to make the best use of this handheld computer.

4-2-1 Searching with a Program

1 "Quick search" with a program

To search for a telephone number with the use of the **[MEMO]** key, we must first find a name (conditional search) and move the display with a cursor control key to obtain the telephone number.

When a record consists of only two items, for example, name and telephone number, it is not so difficult to get data. However, if a record consists of three or four items, it is somewhat tedious to move the display with the **[MEMO]** key or a cursor control key each time you get data.

To process much data more efficiently, we will prepare the "electronic telephone book program" which displays a telephone number immediately when a name is entered.

Input the program correctly, or an error will cause either the program to stop with the message **ERR** during execution or incorrect data to be displayed. A semicolon (;), colon (:), comma (,), or quotation mark (") has its own meaning. Enter them carefully.

2 Program input

Specify the WRT mode by pressing **[MODE]** **[1]** to input the program. The display will be as shown.

```

WRT DEG 7459
P 0123456789

```

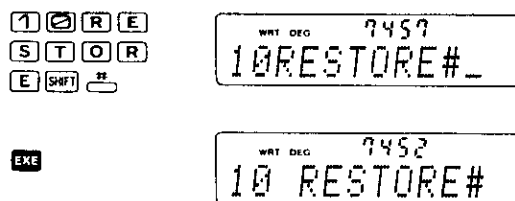
We are now ready for inputting the program. For the program structure and commands, see "CHAPTER 3 BASIC PROGRAMMING" and "CHAPTER 5 COMMAND REFERENCE". Let's input the program.

Press the **[AC]** key to clear the display (You can also input the program without pressing the **[AC]** key).

First, enter line number 10.

Next, enter **[R]** **[E]** **[S]** **[T]** **[O]** **[R]** **[E]**, press the **[SHIFT]** and **[E]** keys. In the shift mode, **[E]** is entered as #.

Make sure that there is no error, and press the **[EXE]** key.



Next, enter line 20. After entering one line, be sure to compare the display and the program listing before pressing the **EXE** key.

When the **EXE** key is pressed, the program line is stored in the computer. You can also check whether or not the program line is stored in the computer by the number of remaining steps displayed.

Program listing

```

10 RESTORE#
20 READ# $
30 A$= MID$(1,7)
40 PRINT A$;
50 INPUT $
60 IF $="" THEN 40
70 FOR I=0 TO 4
80 D$(I)= MID$(1,7)
)
90 $= MID$(8)
100 NEXT I
110 J=0
120 RESTORE# D$+E$+
F$+G$+H$,1,170
130 READ# $,$
140 J=1
150 PRINT $
160 GOTO 120
170 IF J=0 THEN PRI
NT "No Data"
180 END

```

Number of steps: 160

3 Correction

• Correction before pressing the **EXE** key:

Before pressing the **EXE** key, the cursor blinks. Move the cursor with a cursor control key to the character to be corrected, enter the correct character, and press the **EXE** key.

• Correction after pressing the **EXE** key:

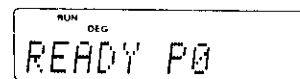
If you find an error after pressing the **EXE** key, enter **L I S T**, followed by the line number which has an error, and press the **EXE** key. The line will be displayed. Correct the error and press the **EXE** key again.

• Correction without using the **LIST** command:

Re-enter the line number to be corrected and then the correct program line, and press the **EXE** key. The new line will replace the incorrect line. If you enter only a line number and press the **EXE** key, the program on that line is deleted.

4 Program execution

A program is executed in the RUN mode. Press **MODE** **☐** to specify the RUN mode. **READY P0** will be displayed and the computer will be ready for program execution.



A program can be executed in one of the following two methods:

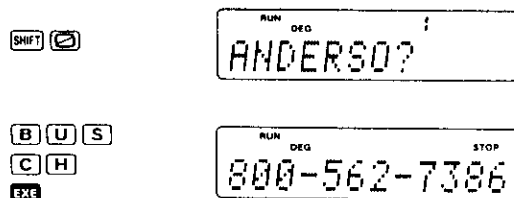
In the first method, enter **R U N** or **SHIFT** **RUN** and press the **EXE** key.

In the second method, press the **SHIFT** key and a program area No. In this example, press **☐** because the program is in program area P0.

Execute the program in one of the two methods.

When a program input error is found, the computer displays **ERR2** and the line number where the error was found. **ERR2** means a syntax error, e.g., misspelled command. Press the **AC** key to cancel the error condition, specify the WRT mode, correct the error, and specify the RUN mode again.

Now, let's execute the program. This program immediately displays the telephone number of the specified person. Press **SHIFT** **☐**. **ANDERSO?** will be displayed. Enter JACKSON and press the **EXE** key. JACKSON's telephone number will be displayed. Next, check BUSCH's telephone number. Press **SHIFT** and **☐** again. **ANDERSO?** will be displayed. Ignore the display. Enter BUSCH and press the **EXE** key.

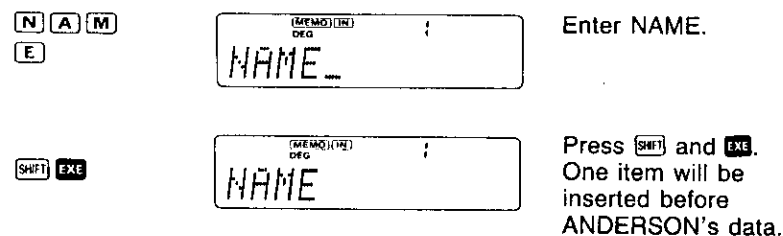
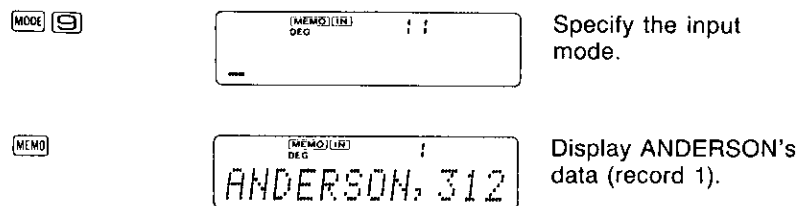


If the computer displays the message "NAME?", the program would be very easy to use. In addition, we cannot check ANDERSON's telephone number with this program. Enter ANDERSON. The computer will answer "No Data". ANDERSON's telephone number is in memory. Why can't we get ANDERSON's telephone number?

This was caused by lines 10 to 40. The commands on these lines read the first item of record 1 and display the first seven characters. Since record 1 contains ANDERSON's data, **ANDERSO?** is displayed.

We can correct this easily.

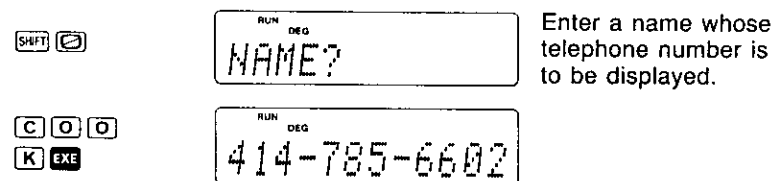
All we have to do is insert "NAME" into record 1. Let's correct the program.



We have finished program correction. Specify the RUN mode and execute the program. When the computer asks "NAME?", enter the name whose telephone number is to be checked and press the **EXE** key. The telephone number will be displayed immediately.

LIST#	7 JACKSON, 514-364
1 NAME	-8839
2 ANDERSON, 312-78	8 JOHNSON, 214-384
0-1234	-3569
3 BUSCH, 800-562-7	9 JONES, 713-461-7
386	132
4 COLLINS, 314-837	10 SMITH, 615-259-2
-1007	935
5 COOK, 414-785-66	11 WILSON, 403-352-
02	6221
6 GATES, 612-536-4	
321	

The stored telephone numbers are shown above. **NAME** appears on line 1. To find a telephone number, specify **SHIFT** **☐** in the RUN mode.



You can enter only the initial letter of a name.

4-3. EXAMPLES OF DATA BANK PROGRAM

4-3. EXAMPLES OF DATA BANK PROGRAM

CASIO's handheld computer with the DATA BANK function is being used in many fields by businessmen or hobbyists.

To make you understand how powerful it is, we will show you some examples in this chapter.

Mr. Kennedy is a businessman working for a trading company. As a manager, he is very busy. To make best use of time, he is trying to find a way to use limited time most efficiently.

Let's take a look at some programs he developed.

4-3-1 Converting Weights and Measures

It is rather tiresome to convert weights and measures (pound, ounce, mile, yard, etc).

To get accurate values, you must use a calculator.

Mr. Kennedy also finds it very tedious to convert values. He has solved this problem with the DATA BANK function. That is, he uses a conversion program to calculate values.

1 Inputting the program

Press **SHIFT** **MODE** in the WRT mode to specify the program area P0.

```
P0
10 INPUT "Number",
  $
20 IF $="" THEN 10
30 D= VAL($ )
40 INPUT "Unit (from)", $
50 IF $="" THEN 40
60 RESTORE#
70 RESTORE# $+ ",",
  1,40
80 READ# A
```

```
90 INPUT "Unit (to)", $
100 IF $="" THEN 90
110 RESTORE#
120 RESTORE# $+ ",",
  1,90
130 READ# B, $
140 C=D*B/A
150 PRINT C, " ("; $; "
  "
160 END
```

Number of steps: 174

2 Entering data

When using this program, make sure that the previous data have been deleted before entering conversion data.

Enter data in the input mode in accordance with the conversion unit table.

MODE **MODE**

1,mile,MILE, **EXE**

1.60931,km,KM, **EXE**

⋮

```
LIST #
1 1,mile,MILE,
2 1.60931,km,KM,
3 1609.31,m,M,
4 63360,inch,INCH
,
5 5280,ft,FT,
6 1760,yd,YD,
7 1,kg,KG,
8 1000,g,G,
9 35.2739,oz,OZ,
10 2.20462,lb,LB,
```

3 Executing the program

Let's convert 3,500 feet to meters.

OPERATION	DISPLAY	
MODE MODE		
SHIFT MODE	Number?	Enter a number to be converted.
3500 EXE	Unit (from)?	Enter the original unit (feet).
FT EXE	Unit (to)?	Enter the target unit (meter).
M EXE	1066.777462	
EXE	[m]	

4 Application

This program can be used for many types of conversion by storing various units in DATA BANK. For example, it is used for converting other weights, measures, and exchange rates.

• Adding exchange rates

If one dollar is 150 yen, enter "1,USD," and "150,JYE,". If one pound is 240 yen, enter "0.625 (150/240), STG,". Enter the exchange rates of franc and mark, with FFR and DMK appended respectively. Now, you can perform exchange rate conversion very easily.

Data to be added

```
11 1,USD,
12 150,JYE,
13 0.625,STG,
```

5 Example

The program shown below does not use the DATA BANK function. If the DATA BANK function is not available, conversion data must be written in the program with the DATA statement.

```
10 INPUT "Number",
  $
20 IF $="" THEN EN
  D
30 N=VAL($)
40 INPUT "Unit: (fr
  enc), $
50 IF $="" THEN 10
60 A$=MID$(1,7)
70 INPUT "Unit: (to
  ), $
80 IF $="" THEN 40
90 B$=MID$(1,7)
```

```
100 RESTORE :J=0
110 READ $,E
120 IF $="END" THEN
  160
130 IF A$=$ THEN C=
  E:J=J+1
140 IF B$=$ THEN D=
  E:J=J+1
150 GOTO 110
160 IF J=2 THEN PRI
  NT "Not Availab
  le": GOTO 40
170 N=N*D/C
```

```
180 PRINT N
190 GOTO 10
300 DATA MILE,1
310 DATA KM,1.60931
320 DATA M,1609.31
330 DATA INCH,63360
340 DATA FT,5280
350 DATA YD,1760
360 DATA KG,1
370 DATA G,1000
380 DATA OZ,35.2739
390 DATA LB,2.20462
400 DATA END,0
```

Number of steps: 373

4-3-2 Matrix Calculation

Mr. Kennedy has various types of data. He must analyze and collect data and produce a report. The number of columns or lines of a report is not fixed. It is rather tedious to calculate the vertical and horizontal calculation for each type of data.

The Matrix Calculation Program automatically calculates the vertical and horizontal totals by simply specifying the number of columns and lines and by entering data.

1 Inputting the program

Input the program in P0.

```
10 INPUT "(m X n)"
  ,A,B
20 FOR C=1 TO A
30 FOR D=1 TO B
40 PRINT "(C,D):"X"
  10:"":
50 INPUT E
60 $="M"+STR$(D)+
  ", "+STR$(E)
70 IF A=C THEN $=$
  +".0"
80 WRITE# $
90 NEXT D
100 WRITE# "M"+STR
  $(C)+".0"
110 NEXT C
120 C=B-20: IF C>0
  THEN DEFN C
130 FOR C=1 TO B
140 F(C)=0
150 NEXT C
160 RESTORE#
170 FOR C=1 TO A
180 F=0
190 FOR D=1 TO B
```

```
200 READ# $,E
210 F(C)=F(C)+E
220 IF A=C THEN WRI
  TE# F(C)
230 F=F+E
240 NEXT D
250 READ# $: WRITE#
  F
260 NEXT C
270 RESTORE#
280 FOR C=1 TO A
290 RESTORE# "M"
300 READ# $,E
310 PRINT $: "=";E
320 NEXT C
330 RESTORE#
340 IF A>1 THEN RES
  TORE# "M"+STR$
  (A-1)
350 READ# $,E
360 FOR C=1 TO B
370 READ# $,E,E
380 PRINT $: "=";E
390 NEXT C
400 END
```

Number of steps: 375

2 Executing the program

Let's calculate the vertical and horizontal totals of the table shown below.

$\begin{matrix} m \\ n \end{matrix}$	Product A	Product B	Product C	Product D	Horizontal total
Branch A	5329	4280	3602	2310	
Branch B	2682	6313	4203	1128	
Branch C	5113	3229	5176	965	
Vertical total					

OPERATION	DISPLAY	
SHIFT ☐	(m × n)?	m: No. of columns n: No. of lines Enter 4 as the number of columns.
4 EXE	?	Enter 3 as the number of lines.
3 EXE	(1 × 1)?	Enter data in column 1, line 1.
5329 EXE	(1 × 2)?	Enter data in column 1, line 2.
2682 EXE	(1 × 3)?	Enter data in column 1, line 3.
5113 EXE	(2 × 1)?	Enter data in column 2, line 1.
4280 EXE	(2 × 2)?	
⋮	⋮	

The left number in parentheses represents a column number, and the right number represents a line number. When data have been entered in all the lines of a column, the program proceeds to the next column.

When you enter 965 in column 4, in line 3, the display will change as shown below.

OPERATION	DISPLAY	
965 EXE	M1 = 13124	Vertical total of column 1
EXE	M2 = 13822	Vertical total of column 2
EXE	M3 = 12981	Vertical total of column 3
EXE	M4 = 4403	Vertical total of column 4
EXE	N1 = 15521	Horizontal total of line 1
EXE	N2 = 14326	Horizontal total of line 2
EXE	N3 = 14483	Horizontal total of line 3

As shown above, the vertical and horizontal totals will be calculated.

3 Use of data

Since data entered during matrix calculation are stored in the DATA BANK, they can be checked in the input mode. In addition, you can use a printer (FP-12S, FP-40 connected with SB-42 interface pack) to print data as follows:

MODE **☐**
LIST **SHIFT** **☐**
EXE

LIST #
1 N1,5329
2 N2,2682
3 N3,5113
4 M1,13124
5 N1,4280
6 N2,6313
7 N3,3229
8 M2,13822

9 N1,3602
10 N2,4203
11 N3,5176
12 M3,12981
13 N1,2310,15521
14 N2,1128,14326
15 N3,965,14483
16 M4,4403

Be sure to delete data by entering NEW # in the WRT mode before executing the program when new data are entered.

4-3-3 Product Code Check List

Recently, more and more products have their codes. It is difficult to associate product codes with their names although you can associate product names with products. Mr. Kennedy uses the Product Code Check Program to convert from codes to product names and vice versa. This program eliminates the need for checking product codes when preparing slips.

1 Inputting the programs

Input the program, which is used to enter codes and product names, in program area P0. Input the program, which is used to convert codes to product names or vice versa, in program area P1.

```
P0
10 INPUT "Code", $
20 IF $="" THEN EN
  D
30 FOR I=0 TO 3
40 A$(I)= MID$(I*7
  +1,7)
50 NEXT I
60 INPUT "Product
  Name", $
70 IF $="" THEN #60
80 RESTORE# : REST
  ORE# "END",0,9
  0
90 WRITE# A$+B$+C$
  +D$+,"+++",
100 WRITE# "END"
110 GOTO 10
```

Number of steps: 146

```
P1
10 INPUT "Key Word
  ", $
20 IF $="" THEN EN
  D
30 FOR I=0 TO 3
40 A$(I)= MID$(I*7
  +1,7)
50 NEXT I
60 J=0
70 RESTORE#
80 RESTORE# A$+B$+
  C$+D$+,"+",1,200
  0
90 J=1
100 READ# $
110 FOR I=0 TO 3
120 E$(I)= MID$(I*7
  +1,7)
130 NEXT I
140 READ# $
150 IF $+A$+B$+C$+D
  $ THEN 170
160 $=E$+F$+G$+H$
170 PRINT $
180 GOTO 80
190 IF J=0 THEN PRI
  NT "No Data"
200 GOTO 10
```

Number of steps: 217

2 Entering data

OPERATION	DISPLAY
SHIFT 	Code? Enter a code.
6020S-FD1 	Product Name? Enter a product name.
5.25-INCH FDD UNIT 	Code?
6024S-FD2 	Product Name?
8-INCH FDD UNIT 	Code?
⋮	⋮

As shown above, codes and product names are repeatedly entered.

```
LIST #
1 6020S-FD1,5.25-
  INCH FDD UNIT,
2 6024S-FD2,8-IN
  CH FDD UNIT,
3 6025S-DK1,DISK
  SHEET,
4 1017PR,GRAPHIC
  PRINTER,
5 6017S-PR,CHARAC
  TER PRINTER,
6 6032S,ADD.RAM 8
  OARD,
7 END
```

Entered data can be deleted in the input mode.

3 Displaying data

You can convert product names to codes, or vice versa.

OPERATION	DISPLAY
SHIFT 1	Key Word? Enter a product name to check its code.
DISK SHEET EXE	6025S-DK1 The code is displayed.
EXE	Key Word? Enter a code to check its product name.
1017PR EXE	GRAPHIC PRINTER The product name is displayed.
EXE	Key Word?

4-3-4 Traveling Expense Calculations

It is tiresome to calculate traveling expenses after a business trip. Usually, you are very busy after a trip and, so, there is no time to do expense calculation. When preparing a detailed expense report, you must check your schedule and receipts.

In this case, the Expense Program is very useful. All you have to do is enter what you do and how much you spend every day during the trip. After the trip, the program instantly gives you information about what you did and how much you spent. Stored data are sorted according to dates, and the daily expenses are automatically calculated.

**You can get print forms if you connect an FP-12S or SB-42 (for FP-40) to the computer.*

1 Inputting the program

Input the data entry program in P0, and the data retrieval program in P1.

```
P0
10 INPUT "Date",J
20 IF J="" THEN EN
  D
30 N= VAL(J)
40 A= INTN:B= INT(
  FRACN*100)
50 IF A<1 THEN IF
  A<12 THEN 70
60 GOTO 10
70 IF B<1 THEN IF
  B<31 THEN 90
80 GOTO 10
90 J= STR$(100+B)
100 A$= STR$(A)+","
  + MID$(Z)
110 INPUT "Memo-1",
  J
120 IF J="" THEN 11
  0
130 FOR Z=0 TO 3
140 B$(Z)= MID$(Z*7
  +1,7)
150 NEXT Z
```

```
160 INPUT "Memo-2",
  J
170 IF J="" THEN 16
  0
180 FOR Z=0 TO 3
190 F$(Z)= MID$(Z*7
  +1,7)
200 NEXT Z
210 INPUT "Amount $
  ",J
220 IF J="" THEN 21
  0
230 Y= INT VAL(J)
240 RESTORE#
250 RESTORE# A$,0.5
  10
260 RESTORE# A$+",T
  OTAL,!"
270 READ# J$,J$,X
280 X=X+Y
290 RESTORE# "",1
300 WRITE# A$,B$+C$
  +D$+E$,F$+G$+H$
  +I$,Y,""
```

```
310 FOR Z=0 TO 7
320 B$(Z)=""
330 NEXT Z
340 B$="TOTAL":F$=""
  !"
350 READ# J$
360 IF J$="END" TH
  EN 350
370 READ# J
380 FOR Z=0 TO 3
390 K$(Z)= MID$(Z*7
  +1,7)
400 NEXT Z
410 READ# J,Y
420 RESTORE# "",1
430 WRITE# A$,B$+C$
  +D$+E$,F$+G$+H$
  +I$,X,""
440 A$=J$
450 FOR Z=0 TO 3
460 B$(Z)=K$(Z)
470 F$(Z)= MID$(Z*7
  +1,7)
480 NEXT Z
```

```

490 X=Y
500 GOTO 350
510 RESTORE# : REST
   ORE# ".END",0,5
   20
520 WRITE# A$+",""+B
   $+C$+D$+E$+",""+
   F$+G$+H$+I$+",""+
   +STR$(Y)+",""+
530 Y=Y
540 GOTO 560
550 RESTORE# : REST
   ORE# ".END"
560 WRITE# A$+","TOT
   AL$+",""+STR$(X)
   +",""+
570 WRITE# ".END"
580 GOTO 10

```

Number of steps: 741

```

P1
10 INPUT "Date", $
20 IF $="" THEN EN
   D
30 N=VAL($)
40 A=INTN:B=INT(
   FRACN*100)
50 IF A=1 THEN IF
   A=12 THEN 70
60 GOTO 10
70 IF B=1 THEN IF
   B=31 THEN 90
80 GOTO 10
90 $=STR$(100+B)
100 A$=STR$(A)+",""+
   +MID$(2)
110 RESTORE#

```

```

120 RESTORE# A$+","T
   OTAL$,",",0,100
130 READ# $, $, X
140 PRINT "$":X
150 GOTO 10
160 PRINT "No Data"
170 GOTO 10

```

Number of steps: 198

2 Entering data

OPERATION	DISPLAY
	Date? Enter the date. Enter the day in two digits and separate it from the month with the  key.
5.02 	Memo-1? Enter Airplane.
AIRPLANE 	Memo-2? Enter Boston-NY.
BOSTON-NY 	Amount \$? Enter the amount.
50 	Date? The first question appears.

Enter data within 30 characters.

In the above example, you entered AIRPLANE in Memo-1, and the traveling section in Memo-2. You may also enter two means of traveling, e.g., TAXI and AIRPLANE, in Memo-1 and Memo-2, and the amount in Amount \$.

This program automatically calculates the total amount of each day and outputs listing as shown in the following example, List 1.

List 1

```

LIST #
1 5.02,AIRPLANE,B
   OSTON-NY,50,
2 5.02,TAXI,TEL,5
   ,
3 5.02,DINNER/DRI
   NK,MR.GARDNER,7
   0,
4 5.02,TOTAL,!,12
   5,
5 5.03,AIRPLANE,M
   Y-BOSTON,50,
6 5.03,TAXI,AIRPO
   RT-HOME,9,
7 5.03,SOUVENIR,M
   ARGALET,7,
8 5.03,TOTAL,!,66
   ,
9 .END

```

List 2

```

LIST #
1 5.02,AIRPLANE,B
   OSTON-NY,50,
2 5.02,TAXI,TEL,5
   ,
3 5.02,DINNER/DRI
   NK,MR.GARDNER,7
   0,
4 5.02,LUNCH,MR.N
   ELSON,10,
5 5.02,TOTAL,!,13
   5,
6 5.03,AIRPLANE,M
   Y-BOSTON,50,
7 5.03,TAXI,AIRPO
   RT-HOME,9,
8 5.03,SOUVENIR,M
   ARGALET,7,
9 5.03,TOTAL,!,66
   ,
10 .END

```

3 Adding data

Assume that you failed to enter the charge of the lunch you had with Mr. Nelson on May 2. You can add the data in the same way when entering data.

OPERATION	DISPLAY
	Date?
5.02 	Memo-1?
LUNCH 	Memo-2?
MR.NELSON 	Amount \$?
10 	Date?

See the listing above, List 2. "LUNCH,MR.NELSON,10" has been added at the end of the data for May 2, and the total amount has been updated.

4 Displaying data

To display data, press **SHIFT** **F1** in the RUN mode.

OPERATION	DISPLAY	
SHIFT F1	Date?	Enter the date. Enter the day in two digits and separate if from the month with the □ key.
5.02 EXE	\$135	The grand total for May 2 is displayed.
EXE	Date?	

If there is no data for the day, **No Data** is displayed. Pressing the **EXE** key again will display the initial message.

5 Application

The Expense Program can be used not only for calculating business trip expenses but also for storing daily expenses or traveling expenses. If you want to use this program to store traveling expenses, change Memo-1 on line 110 to "From", and Memo-2 on line 160 to "To". The program will display a question asking you about a traveling distance.

For example, you can enter BROOKLYN in response to "From?", TIMES SQUARE in response to "To", and 1 in response to "Amount \$?".

A taxi driver can use this program for a daily report.

4-3-5 Sales Data

Mr. Kennedy keeps sales data in chronological order. He keeps order data (item and quantity) and prints the result at the weekend. This program not only streamlines daily business management but also allows him to get weekly and monthly data easily.

1 Inputting the programs

Input the data entry program in P0, and the data retrieval program in P1.

```
P0
10 INPUT "Date",#
20 IF #="" THEN EN
  0
30 N= VAL(#)
40 A= INTN:B= INT(
  FRACN*100)
50 IF A≥1 THEN IF
  A≥12 THEN ?0
60 GOTO 10
70 IF B≥1 THEN IF
  B≥31 THEN ?0
80 GOTO 10
90 #= STR$(B+100)
100 A#= STR$(A)+". "
  + MID$(2)
110 INPUT "Item-1",
  $
120 IF #="" THEN 11
  0
130 FOR Z=0 TO 3
140 B$(Z)= MID$(Z*7
  +1,7)
150 NEXT Z
160 INPUT "Item-2",
  $
170 IF #="" THEN 16
  0
180 FOR Z=0 TO 3
190 F$(Z)= MID$(Z*7
  +1,7)
```

```
200 NEXT Z
210 INPUT "Item-3",
  $
220 IF #="" THEN 21
  0
230 FOR Z=0 TO 3
240 J$(Z)= MID$(Z*7
  +1,7)
250 NEXT Z
260 RESTORE#
270 RESTORE# "",0,3
  00
280 READ# Y$
290 IF Y$="!END" TH
  EN RESTORE# : K
  ESTORE# "!END":
  GOTO 500
300 IF VAL(A$)≥ VAL
  (Y$) THEN READ#
  $,$,$,$: GOTO
  270
310 READ# $
320 FOR Z=0 TO 3
330 N$(Z)= MID$(Z*7
  +1,7)
340 NEXT Z
350 READ# $
360 FOR Z=0 TO 3
370 R$(Z)= MID$(Z*7
  +1,7)
380 NEXT Z
```

```
390 READ# $
400 RESTORE# "",1
410 WRITE# A$,B$+C$
  +D$+E$,F$+G$+H$
  +I$,J$+K$+L$+M$
420 A$=Y$
430 FOR Z=0 TO 3
440 B$(Z)=N$(Z)
450 F$(Z)=R$(Z)
460 J$(Z)= MID$(Z*7
  +1,7)
470 NEXT Z
480 READ# $
490 GOTO 270
500 #=J$+K$+L$+M$
510 WRITE# A$+"," +B
  $+C$+D$+E$+"," +
  F$+G$+H$+I$+"," +
  J$+K$+L$+M$+"," +
  +$+"," + "
520 WRITE# "!END"
530 GOTO 10
```

Number of steps: 674

```

P1
10 INPUT "Key Word
  ",$
20 IF $="" THEN EN
  D
30 FOR I=0 TO 3
40 A$(I)= MID$(I*7
  +1,7)
50 NEXT I
60 RESTORE#
70 J=0
80 RESTORE# A$+B$+
  C$+D$,1,200
90 J=1

```

```

100 READ# $
110 PRINT $
120 READ# $
130 PRINT $
140 READ# $
150 PRINT $
160 READ# $
170 PRINT $
180 READ# $
190 GOTO 80
200 IF J=0 THEN PRI
  NT "No Data"
210 GOTO 10

```

Number of steps: 177

2 Entering data

Enter the following sales data:

June 21	New York	XX-CAR	12
June 21	Los Angeles	YX-CAR/MK2	5
June 21	Boston	YX-CAR	8
June 22	Chicago	XX-CAR	6
June 23	San Francisco	YX-CAR	7

Sales data

```

LIST #
1 6.21,NEW YORK,X
  X-CAR,12,!
2 6.21,LOS ANGELE
  S,YX-CAR/MK2,5,
  !
3 6.21,BOSTON,YX-
  CAR,8,!
4 6.22,CHICAGO,XX
  -CAR,6,!
5 6.23,SAN FRANCIS
  CO,YX-CAR,7,!
6 6.24,NEW ORLEAN
  S,XX-CAR,9,!
7 6.25,DALLAS,XX-
  CAR,15,!
8 6.28,SEATTLE,XX
  -CAR,4,!
9 !END

```

OPERATION	DISPLAY
SHIFT	Date?
	Enter the date. Enter the day in two digits and separate it from the month with the key.
6.21	Item-1?
NEW YORK	Item-2?
XX-CAR	Item-3?
12	Date?
	The initial message appears after the data for item-3 is entered.

Enter data within 30 characters. Data are automatically sorted into chronological order.

3 Displaying data

Entered data can be searched for with any item you want. For example, let's check the sales result of XX-CAR.

OPERATION	DISPLAY
SHIFT	Key Word? Enter a keyword with which you search data.
XX-CAR	6.21
	NEW YORK
	XX-CAR
	12
	6.22
	CHICAGO
	XX-CAR
	6
	6.24
	NEW ORLEANS
	XX-CAR
	9

As shown above, all the data associated with the keyword "XX-CAR" is displayed.

Similarly, if you enter the keyword "LOS ANGELES", the sales data of the Los Angeles branch will be displayed in chronological order. When no corresponding data is left, the initial message "Key Word?" is displayed.

COMMAND REFERENCE

[illegible]

* The following descriptions apply symbols and terms frequently used in the syntax.

- { $\begin{smallmatrix} \times & \times & \times & \times \\ \bigcirc & \bigcirc & \bigcirc & \bigcirc \end{smallmatrix}$ } One of the elements inside { } must be selected.
- [$\begin{smallmatrix} \bigcirc & \bigcirc & \bigcirc & \bigcirc \end{smallmatrix}$] The element inside [] can be omitted
- $\begin{smallmatrix} \bigcirc & \bigcirc & \bigcirc & \bigcirc \\ & & & * \end{smallmatrix}$ The element with * on the top right can be repeatedly used.
- Numerical expression
Numerical value, calculation expression, and numerical variable such as 10, 2+3, A, \$*Q.
- Character expression
Character constant, character variable, and character expression such as "ABC", X\$, N\$+M\$.
- Parameter An element that accompanies a command.
- Ⓟ Can only be executed in a program.
- Ⓜ Can only be executed manually.
- Ⓐ Can be executed both manually and in a program.
- Ⓜ Function instruction that can be executed both manually and in a program.

〈Example〉 DATA [data] [, [data]]*

Since all data are provided with a bracket [], it will also be possible to write "DATA" only. Since [, [data]] is provided with []*, this element can be written repeatedly. This can therefore be written "DATA data, data, ..." If we omit the first [data], this can also be written "DATA, data, data, ..."

**GOTO { Line No.
program area No. }**

There are two different ways to write this statement as shown below.

- (1) GOTO line No.
- (2) GOTO # program area No.

Function

Program erase. Erases programs and variables.

Parameter

When ALL is specified, all P0 ~ P9 programs and variables are erased.

Explanation

- (1) If ALL is not specified, the program in the presently specified program area is erased. Variables are not erased.
 - (2) If ALL is specified, the programs in all program areas and variables are erased. The DEFM setting is released and the number of memories is initialized to 26.
 - (3) Cannot be executed while a password is specified.
 - (4) Cannot be used in a program.
 - (5) Can only be executed in the WRT mode.
- * NEW ALL can be abbreviated as NEW A.

Example

MODE 1 NEW EXE

RUN

[Execution start line]

line No.



Function

Program execution.

Parameter

When a line is specified, execution starts from the line.

Explanation

- (1) Executes a program from a specified line (when the line number is omitted, execution starts from the beginning of the program).
- (2) When a specified line number does not exist, execution starts from the line with the closest larger number.
- (3) Variables are not cleared.

Example

```
10 PRINT "LINE 10"  
20 PRINT "LINE 20"  
30 END
```

```
RUN EXE  
RUN 20 EXE
```

LINE 10
LINE 20

LIST

[{line No.}]
ALL



Function

Displays the content of a program.

Parameter

Line No.: No. of the first line to be displayed.

ALL: Displays the content of all P0—P9 programs sequentially.

Explanation

I. RUN mode

- (1) Sequentially displays the content of a program from a line number if it is specified, or from the beginning if it is omitted.
- (2) Since the content of a program is automatically displayed sequentially, press the **STOP** key to stop this. Press the **EXE** key to display the next line and after.
- (3) In the PRINT mode (when "PRT" is displayed) with a printer connected, the display is not stopped but is made sequentially at high speed.

II. WRT mode

- (1) Displays the content of a program from a line number if it is specified, and from the beginning if it is omitted.
 - (2) Since each line is displayed for edit in the WRT mode, if edit is not required, press the **EXE** key to advance to the next line. Also, if the **STOP** key is pressed before the **EXE** key, the previous line is displayed.
- When ALL is specified, the content of all P0—P9 programs are sequentially displayed. In this case they are sequentially advanced even in the WRT mode, so edit cannot be performed.
 - This command cannot be used while a password is specified.
- * LIST ALL can be abbreviated as LIST A.

Example

```
LIST EXE  
LIST 30 EXE
```

PASS

"Password"
Character string



Function

Specifies or releases a password.

Parameter

Password: String with 1—8 characters.

Explanation

- (1) If this command is executed when a password is not specified, a password is specified for all program areas (P0—P9).
- (2) If this command is executed while a password is specified, this password is released only when entering the corresponding password. When passwords do not correspond, a protect error (ERR8) occurs.
- (3) A password consists of a 1—8 character string in which spaces, alphabetical characters, numerals, special symbols, etc. can be used. However, (") cannot be used.
- (4) While a password is specified, commands such as LIST, LIST ALL, LIST #, NEW, NEW ALL and NEW # cannot be used. Also no writing (WRT mode) can be made; if it is attempted, an error (ERR8) occurs.
- (5) Cannot be used in a program.
- (6) A password can be maintained while the power switch is off.
- (7) If a program is stored on a cassette tape by a SAVE or SAVE ALL command while a password is specified, this password is also stored. When a program with a password attached is loaded from a cassette tape by a LOAD or LOAD ALL command, the password is also loaded. Also, when a currently specified password in the mainframe and the password of a program loaded from a cassette tape are different, the program cannot be loaded from a cassette tape (ERR8).

Precaution

If a password was forgotten after it was specified, press the ALL RESET button on the back of the computer and clears all the programs and memory.

Example PASS "CASIO" 

SAVE [ALL]

["File name"]
Character string



Function

Stores a program on a cassette tape.

Parameter

ALL: Stores the programs in all the program areas.

File name: String with 1—8 characters. Can be omitted.

Explanation

- (1) When ALL is omitted, the content in the presently specified program area is stored.
 - (2) When ALL is used, the contents of all P0—P9 program areas are stored.
 - (3) When a password is specified, the storing is performed with that password. Therefore, the password is the same as that stored when the program is loaded by the LOAD command.
- * SAVE ALL can be abbreviated as SAVE A.

Example

```
SAVE   
SAVE "CASIO"   
SAVE ALL "PB" 
```

LOAD [ALL] ["File name"]

Character string



Function

Loads a program from a cassette tape.

Parameter

ALL: Loads the programs in all program areas.

File name: String with 1—8 characters. Can be omitted.

Explanation

- (1) When ALL is omitted, a program stored by "SAVE" is read into the presently specified program area.
- (2) When ALL is used, programs stored by "SAVE ALL" are read into the P0—P9 program areas.
- (3) When a program stored with a password attached is loaded from a cassette tape, this password is also loaded.

* Load ALL can be abbreviated as LOAD A.

SAVE and LOAD Relationship

	LOAD	LOAD "File name"	LOAD ALL	LOAD ALL "File name"
SAVE	○	×	×	×
SAVE "File name"	○	○	×	×
SAVE ALL	×	×	○	×
SAVE ALL "File name"	×	×	○	○

* File names are identical. ○... Can be loaded.
×... Cannot be loaded.

VERIFY ["File name"]

Character string



Function

Checks the status of a program and data stored on a cassette tape.

Parameter

File name: String with 1—8 characters. Can be omitted.

Explanation

- (1) When a file name is specified, the file with this name is checked.
- (2) When the file name is omitted, checks the first file that appears on the cassette tape.
- (3) The parity check system is used to check a storing format.

Example

```
VERIFY EXE  
VERIFY "PROG1" EXE
```

CLEAR



Function

Clears all variables.

Explanation

- (1) Clears all variables; all numerical variables are cleared to 0 and all character variables to a null.
- (2) This command can be used both in a program and manually.
- (3) Since control variables are also cleared in a FOR-NEXT loop (see page 158), an error occurs during NEXT statement execution.

* The CLEAR command functions the same as VAC.

END

(P)

Function

Terminates program execution.

Explanation

Since program execution is terminated, the next program is not executed even if it exists.

STOP

(P)

Function

Temporarily suspends program execution.

Explanation

- (1) Temporarily suspends program execution and displays "STOP" after which input waiting occurs.
- (2) After suspension, execution is resumed by pressing the **EXE** key.
- (3) If the **STOP** key is pressed while execution is stopped by a STOP statement, the program area number and line number are displayed.
- (4) During execution suspension by STOP, manual calculations can be performed.

[LET] { Numerical variable = numerical expression }
{ Character variable = character expression }

(P)

Function

Assigns the value of the expression on the right to the variable on the left.

Explanation

- (1) A numerical expression corresponds to a numerical variable, and a character expression corresponds to a character variable.
- (2) LET can be omitted.

Example

```
10 LET X=12
20 LET Y=X↑2+2*X-1
30 PRINT Y
40 A$="CASIO"
50 B$="PB & FX"
60 PRINT A$;B$
70 END
```

REM

Comment
Character string

(P)

Function

Statement that expresses a comment.

Explanation

- (1) Written in a program. Content after REM is treated as comment statement and is therefore not executed.
- (2) When a command to be executed is written on the same line, write a multistatement sign (:) before the REM statement.

Example

```
10 INPUT "R",R
20 S=π*R↑2:REM AREA
30 PRINT S
40 END
```

INPUT [$\frac{\text{"Message statement"}}{\text{Character string}}$], $\frac{\text{variable}}{\text{name}}$ [, [$\frac{\text{"Message statement"}}{\text{Character string}}$], $\frac{\text{variable}}{\text{name}}$] *

(P)

Function

Inputs data from the keyboard to a variable.

Parameter

Message: Character string

Variable name: Numerical variable name or character variable name.

Explanation

- (1) Input data from the keyboard to a specified variable.
- (2) When a message exists, it is displayed followed by "?".
- (3) When there is no message, only "?" is displayed.
- (4) Press the **EXE** key after data input.

- (5) When character data are entered into a numerical variable, an error (ERR2) occurs and data input is requested again by the display of "?" after the **AC** key is pressed. When a numerical expression is entered, the result of this expression is assigned. When one alphabetical character is entered, the value of the variable corresponding to this character is assigned.
- (6) If the **EXE** key only is pressed without inputting the numeric value or numeric expression, the value in the numeric variable will not change.
- (7) If the **EXE** key only is pressed without inputting a character expression when inputting to a character variable, null will be assigned to the character variable.
- (8) If characters exceeding seven characters are input to a character variable, the first seven characters will be significant and eighth character and after will be disregarded.

Example

```
10 INPUT A
20 INPUT "B$=",B$
30 INPUT "C$=",C$, "D$=",D$
```

KEY\$

(P)

Function

A function that enters one character from the keyboard.

Explanation

- (1) The input of only one character is accepted from the keyboard.
 - (2) Numerals, alphabetical characters, and symbols can be input.
 - (3) Since "?" is not displayed and input waiting does not occur, KEY\$ is usually combined with an IF statement.
- * KEY\$ can be abbreviated as KEY.

Example

```
10 PRINT "INPUT<6>";
20 A$=" "
30 K$=KEY$
40 IF K$=" " THEN 30
50 A$=A$+K$
60 IF LEN(A$)<6 THEN 30
70 PRINT A$
80 END
```

* Six characters are accepted from the keyboard.

PRINT [Output element] [{ ; } [Output element]]* (P)

Function

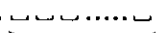
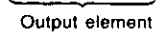
Displays an output element.

Parameter

Output element: Output control function (CSR), numerical expression, character expression.

Explanation

- (1) Displays an output element. When an output control function is added, the element is displayed at the location determined by this function.
- (2) Values are displayed for numerical expressions and character expressions.
- (3) When an output element is a numerical expression, a position for sign (+, -) is placed before the value. However, the + sign is displayed as a blank.

- Character display.....
- Numeral display.....

- (4) When an output element is a numerical expression and the mantissa is more than 10 digits, the 11th digit is rounded off. When an exponent exists besides the mantissa, an exponent sign (E) and a two digit exponent are displayed.
- (5) ",", " and ";" can be used as punctuation between output elements. When ",", " is used, the execution stops (STOP is displayed) after the first output element is displayed, then the next output element is displayed by pressing the **EXE** key. When ";" is used, the next output element is displayed continuously after the first one.
- (6) When no output element is specified (only PRINT is written), the display is cleared and is not stopped.
- (7) The display is not stopped during printing in the print mode (**MODE** .
- (8) The output format of the numerical value can be specified by a SET statement.

Example

```
10 PRINT 1/3
20 PRINT "A=" ; A
30 PRINT "SIN 30" , SIN 30
40 PRINT "END";
50 PRINT
60 END
```

CSR Output location specification

Numerical expression

(F)

Function

Displays an output element from a specified location.

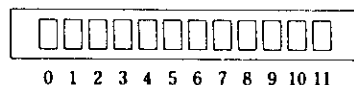
Parameter

Output location specification: Numerical expression. Values below decimal point are discarded.

$$0 \leq \text{specification} < 12$$

Explanation

- (1) Used in a PRINT statement to specify the location of an output element.
- (2) The output location of the left end is 0.



Example

```
10 FOR I=0 TO 11
20 PRINT CSR; "A"; CSR 11-I; "B"
30 NEXT I
40 END
```

- A and B characters are shifted from the left and right respectively each time the **EXE** key is pressed.

GOTO

{ Branching line No.
line No.
program area No.
Number 0 to 9 }

(P)

Function

Unconditionally branches to a specified location.

Parameter

Branching line No.: Line No. from 1 to 9999.

Program area No.: A number from 0 to 9.

Explanation

- (1) Branches to a specified location.
 - (2) When a branching location is a line number, branches to the specified line in the current program area and executes the program. When the branching line number does not exist, an error (ERR4) occurs.
 - (3) When the branching location is a program area number, branches to the specified program area and executes the program from the beginning.
- * A numerical expression can be used for the branching line number and the program area number.

Example

```
10 PRINT "START";
20 GOTO 100
30 PRINT "LINE 30"
40 END
100 PRINT "LINE 100"
110 GOTO 30
```

ON $\frac{\text{Branching condition}}{\text{Numerical expression}}$ **GOTO** $\frac{\text{[Branching location]}}{[, \text{[Branching location]}]}$ ^(P)

★ Branching location $\begin{cases} \text{Branching line No.} \\ \text{\# program area No.} \end{cases}$

Function

Branches to a specified location according to the branching condition.

Parameter

Branching condition: Numerical expression. Values below the decimal point are discarded.

Branching line No.: Line No. from 1 to 9999.

Program area No.: A number from 0 to 9.

Explanation

- (1) Branches according to the integer part of the value in a branching condition expression. Branching locations are allocated sequentially according to

ON A GOTO $\frac{100}{A=1}, \frac{200}{A=2}, \frac{300}{A=3}, \dots$

- (2) When the value of the expression is smaller than 1, or when an appropriate branching location does not exist, the next statement is executed without branching.
- (3) As many branching locations that can fit on one line can be written.

Example

```
10 INPUT A
20 ON A GOTO 100,200,300
30 PRINT "OTHER"
40 GOTO 10
100 PRINT "LINE 100":GOTO 10
200 PRINT "LINE 200":GOTO 10
300 PRINT "LINE 300":GOTO 10
```

- When 1—3 is entered, branchings to 100—300 are performed respectively, otherwise "OTHER" is displayed.

IF $\frac{\text{Branching condition}}{\text{Conditional expression}}$ **THEN** $\frac{\{\text{Statement [: statement]}\}}{\text{Branching location}}$ ^(P)

★ Branching location $\begin{cases} \text{Branching line No.} \\ \text{\# program area No.} \end{cases}$

Function

When a branching condition is true, the statements after THEN are executed. Also, when a statement after THEN is a branching location, branching is performed.

Parameter

Branching condition: Conditional expression

Branching line No.: Line No. from 1 to 9999.

Program area No.: A number from 0 to 9.

Explanation

- (1) When the branching condition is true, the statements after THEN are executed or branching is performed.
- (2) When the branching condition is false, the next line is executed.
- (3) The branching condition is tested by a conditional expression ($=, \neq, <, >, \leq, \geq$).
 - $=$ The item on the left is equal to the item on the right.
 - $\neq$ The item on the left is not equal to the item on the right.
 - $<$ The item on the right is larger than that on the left.
 - $>$ The item on the right is smaller than that on the left.
 - $\leq$ The item on the right is larger than or equal to that on the left.
 - $\geq$ The item on the right is smaller than or equal to that on the left.
- (4) When two or more branching conditions exist, several IF-THEN statements can be written sequentially.

IF — THEN IF — THEN

- * When a statement exists after THEN, ";" can be used instead of THEN.

Example

```
10 N=6
20 PRINT CSR N; "↑";
30 K$=KEY$
40 IF K$="4" THEN N=N-1:IF N<0 THEN N=0
50 IF K$="6" THEN N=N+1:IF N>11 THEN N=11
60 PRINT
70 GOTO 20
```

- "↑" is shifted to the left when the $\leftarrow$ key is pressed and is shifted to the right when the $\rightarrow$ key is pressed.

FOR $\frac{\text{Control variable name}}{\text{Numerical expression}} = \frac{\text{Initial value}}{\text{Numerical expression}}$ **TO** $\frac{\text{Final value}}{\text{Numerical expression}}$ (P)
[STEP $\frac{\text{Increment}}{\text{Numerical expression}}$ **]** **NEXT** $\frac{\text{Control variable name}}{\text{Numerical expression}}$

Function

Repeats process contained between FOR and NEXT statements a number of times specified by the control variable. The value of this variable is changed, from the initial to the final one, by the increment for each repetition of the process.

Parameter

Control variable name: Simple variable name.
 An array variable can not be used.
 Initial value: Numerical expression
 Final value: Numerical expression
 Increment: Numerical expression
 The value 1 is taken in default of this.

Explanation

- (1) Repeats process contained between FOR and NEXT statements a number of times specified by the control variable. The value of this variable is changed, from the initial to the final one, by the increment for each repetition of the process. When the value of the control variable exceeds the final value, repetition is terminated.
- (2) When the initial value is larger than the final value, the execution between FOR-NEXT is performed only once.
- (3) A negative number can be used for an increment.
- (4) A NEXT statement must always correspond to a FOR statement and must be written after it.
- (5) FOR-NEXT loops can have the following nested structure.

```
10 FOR I=1 TO 10
20 FOR J=11 TO 20
30 PRINT I;" ";J
40 NEXT J
50 NEXT I
60 END
```

- (6) Nesting can be performed with up to 4 levels.

- (7) When a FOR-NEXT loop is terminated, the value of the control variable exceeds the final value by the value of the increment.
- (8) A branching out of a FOR-NEXT loop can be performed. If branching inside a FOR-NEXT loop by an IF statement or GOTO statement is attempted, an error occurs.

GOSUB

$\left\{ \begin{array}{l} \text{Branching line No.} \\ \text{Line No.} \\ \text{\# program area No.} \\ \text{A character from 0 to 9} \end{array} \right\}$ (P)

Function

Performs a branching to a specified subroutine.

Parameter

Branching line No.: Line No. from 1 to 9999.
 Program area No.: A character from 0 to 9.

Explanation

- (1) Performs a branching to a subroutine. A return from this subroutine is performed by executing RETURN.
- (2) To make a subroutine inside a subroutine is called nesting which can be performed with up to 8 levels.
- (3) Return to the statement next to the GOSUB statement is performed by RETURN.
- (4) Return to the main routine cannot be performed by an IF statement or GOTO statement. Therefore, be sure to perform return by a RETURN statement.
- (5) When the branching line No. does not exist, an error (ERR4) occurs.
 * A numerical expression can also be used for a branching line number and a program area number.

Example

```

10 PRINT "MAIN 10"
20 GOSUB 100
30 PRINT "MAIN 30"
40 END
100 PRINT "SUB 100"
110 GOSUB 200
120 RETURN
200 PRINT "SUB 200"
210 RETURN

```

RETURN

(P)

Function

Provides a return from the subroutine to the main program.

Explanation

Returns to a statement located just after the statement which called the subroutine.

ON Branching condition
Numerical expression

GOSUB [Branching location]
[, [Branching location]]*

(P)

★ Branching location { Branching line No.
program area No.

Function

Branches to a subroutine according to a branching condition.

Parameter

Branching condition: Numerical expression.

Values below the decimal point are discarded.

Branching line No.: Line No. from 1 to 9999.

Program area No.: A character from 0 to 9.

Explanation

- (1) Performs a subroutine branching by the integer part of the value in a branching condition expression. Branching locations are allocated sequentially according to the value of the expression.

ON B GOSUB 1000, 2000, 3000.....
 B 1 B 2 B 3

- (2) When the value of the expression is smaller than 1 or an appropriate branching location does not exist, the next statement is executed without branching.
- (3) As many branching locations as can fit in one line can be written.

Example

```

10 INPUT A
20 ON A GOSUB 100,200,300
30 GOTO 10
100 PRINT "SUB 100":RETURN
200 PRINT "SUB 200":RETURN
300 PRINT "SUB 300":RETURN

```

- When 1—3 is entered, a branching to the corresponding subroutine occurs.

DATA

[data] [, [data]]*
Constant Constant

(P)

Function

Stores data.

Parameter

Data: Character constant or numerical constant.

Explanation

- (1) Used to write data that is read by a READ statement.
- (2) Plural data can be written by punctuation with " , ".
- (3) If only a DATA statement is executed without a READ statement, no function is performed.
- (4) When a character constant includes " , " , place it inside " " .
DATA ABC, DEF, "GHI,JKL", ...
 1st 2nd 3rd
- (5) When data is omitted, a character string with a length of 0 is taken by default.

DATA A, ,B → DATA A,**,B

DATA , → DATA **,**

DATA → DATA **

READ

Variable name [, [variable name]]*

(P)

Function

Reads the content of a DATA statement.

Parameter

Variable name: Numerical variable or character variable. An array variable can be used.

Explanation

- (1) Allocates data in the currently specified DATA statement sequentially to a specified variable.
- (2) Only numerical type data can be read for a numerical variable.
- (3) Data in DATA statements are read sequentially with the smallest line number first, and sequentially from the beginning in a statement.
- (4) After the necessary data are read by a READ statement, the following data are read by the next READ statement.
- (5) The first data in the program area where a READ statement exists is read by the first execution of this statement after which data in the program area at that time are read sequentially.
- (6) The specification of data to be read can be changed by a RESTORE statement.
- (7) When the number of data in a DATA statement is smaller than the number of variables in a READ statement, an error (ERR4) occurs.
- (8) When a space exists at the beginning of data, it is skipped.

Example

```
10 DATA 1,2,3
20 READ A,B
30 PRINT A;B
40 DATA 4,5
50 READ C,D,E
60 PRINT C;D;E
70 END
```

- Reads data sequentially from a DATA statement and displays them.

RESTORE

[Line No.]
Numerical expression

(P)

Function

Specifies the location of data to be read by a READ statement.

Parameter

Line No.: Numerical expression. Values below the decimal point are discarded.

$1 \leq \text{line No.} \leq 9999$

Explanation

- (1) Specifies a DATA statement where data to be read by a READ statement exist.
- (2) When a line number is omitted, the data specification is cancelled. After this, the first data in the program area where a READ statement exists are specified and read by the first READ statement that is executed.
- (3) When a line number of the program area is specified by a RESTORE statement, data of the DATA statement with this line number are read sequentially by the READ statement.
- (4) When a specified line number does not exist or a DATA statement does not exist on a specified line number and after, an error (ERR4) occurs.

Example

```
10 DATA 1,2,3
20 DATA 4,5
30 READ A,B,C,D,E
40 RESTORE 10
50 READ F,G
60 RESTORE 20
70 READ H,I
80 PRINT A;B;C;D;E;F;G;H;I
90 END
```

PUT

[" File name "] variable 1 [, Variable 2]*
Character string

Ⓐ

Function

Stores data on a cassette tape.

Parameter

File name: A string with 1—8 characters. Can be omitted.
Variable 1, variable 2: Specification of the variable to be stored.

Explanation

- (1) Stores the contents of variables on a cassette tape.
- (2) Variable specifications are written as follows.

PUT A Content of variable A.
PUT A,Z Content of variables A—Z.
PUT A,A (50) Content of variables A—A(50).
PUT \$,D,W Content of the exclusive character variable \$
and of variables D—W.

When the content of the exclusive character variable \$ must be stored,
write \$ first.

- (3) Can be executed both manually and in a program.

GET

[" File name "] variable 1 [, Variable 2]*
Character string

Ⓐ

Function

Loads data stored on a cassette tape into a variable.

Parameter

File name: A string with 1—8 characters. Can be omitted.
Variable 1, variable 2: Specification of the variable to be loaded.

Explanation

- (1) Loads data stored on a cassette tape into a specified variable.
- (2) Variable specifications are written as follows.

GET A Loads in variable A.
GET A,Z Loads in variables A—Z.
GET A,A (50) Loads in variables A—A(50).
GET \$,D,W Loads in the exclusive character variables \$,
and in variables D—W.

- (3) A variable name stored by PUT can be different from the name read by GET.
- (4) When the number of stored data is smaller than the number of variables to be loaded, only the data are loaded sequentially in the first variables.
- (5) It can be executed both manually and in a program.

BEEP

[{ 0 }]
[{ 1 }]

Ⓐ

Function

Generates a beep sound.

Parameter

0: Low sound
1: High sound
0 is taken by default.

Explanation

- (1) Generates a high or low beep sound.
- (2) Can also be used manually.

Example

```
10 $="ABCDEFGHIJKLMNPOQRSTUVWXYZ":N=0
20 FOR I=1 TO 10
30 A$=MID$(RAN#*26+1,1)
40 PRINT CSR4;"<":A$;">";
50 FOR J=1 TO 30
60 K$=KEY$:IF K$="" THEN 80
70 NEXT J
80 IF K$=A$ THEN BEEP 1:N=N+1:GOTO 100
90 BEEP 0
100 PRINT:NEXT I
110 PRINT N;
120 IF N>10 THEN END
130 FOR I=1 TO 10
140 BEEP 0:BEEP 1
150 NEXT I
```

- Press the alphabetical keys that correspond to the displayed characters.

DEFM

[Size of memory expansion]
Numerical expression

Ⓐ

Function

Provides memory expansion.

Parameter

Size of memory expansion: Numerical expression. Values below the decimal point are discarded.
Can be omitted.

$$0 \leq \text{Size of memory expansion} \leq 956$$

Explanation

- (1) Expands the memories (variable area).
- (2) An arbitrary number can be specified according to the remaining number of program steps.
- (3) Since 8 steps are required for each memory expansion, the number of remaining steps is reduced.
- (4) When the size of memory expansion is omitted, the number of currently specified memories is displayed.
- (5) It can be executed both manually and in a program. When it is manually executed, the status (number of expanded memories + 26 basic memories) is displayed. When executed by writing it in a program, the status is not displayed.
- (6) When an attempt is made to perform expansion larger than the number of remaining program steps, an error (ERR1) occurs.
- (7) Specify DEFM 0 to cancel the memory expansion and to return to the 26 basic memories.

Example

```
DEFM 10 [EXE]  ***VAR:36
DEFM [EXE]      ***VAR:36

10 DEFM 10
20 FOR I=1 TO 10
30 INPUT Z(I)
40 NEXT I
  ⋮
```

MODE

Numerical expression

Ⓟ

Function

Sets the state of the computer.

Parameter

Numerical expression: Values below the decimal point are discarded.
 $4 \leq \text{numerical expression} < 9$

Explanation

- (1) Sets the angle unit, print mode or releases this mode depending on the numerical expression used.
- (2) Settings are as follows.
MODE4 Sets the angle unit to degrees.
MODE5 Sets the angle unit to radians.
MODE6 Sets the angle unit to grades.
MODE7 Displays "PRT" and sets the print mode.
MODE8 Releases the print mode.
- (3) Same setting as by the **MODE** key. However, the RUN mode and WRT mode cannot be set using this command. Also, input cannot be performed with the **MODE** key, but by pressing the **[M][O][D][E]** keys.

Example

```
10 MODE 4
20 A=SIN 30
30 MODE 7
40 PRINT A
50 MODE 8
60 END
```

SET

 $\begin{Bmatrix} Fn \\ En \\ N \end{Bmatrix}$

(A)

★ n is an integer from 0 to 9.

Function

Specifies the output format for numerical data.

Parameter

Fn : Specifies the number of decimal places.
 En : Specifies the number of significant digits.
 N : Releases a specification.

Explanation

- (1) Specifies the number of decimal places or significant digits.
- (2) For specifying the number of decimal places (Fn), a value from 0 to 9 is used.
- (3) For specifying the number of significant digits (En), a value from 0 to 9 is used. Also "SET E0" indicates a 10-digit specification.
- (4) Both specifications are released by "SET N".
- (5) It can be executed both manually and in a program.

Example

```
10 INPUT N
20 SET F5:PRINT N
30 SET E5:PRINT N
40 SET N:GOTO 10
```

CHARACTER FUNCTIONS

LEN

(Simple character variable)

(F)

Function

Gives the length of the character string in a simple character variable.

Parameter

Simple character variable: An array variable can not be used.

Explanation

- (1) Counts the number of characters in a simple variable.
- (2) The character variable used is a simple character variable (A\$, Y\$, etc.); an array character variable (B\$ (3), etc.) cannot be used.

Example

```
10 INPUT A$
20 PRINT LEN(A$)
30 GOTO 10
```

MID\$

(Location [, Number of characters])
Numerical expression Numerical expression

Ⓕ

Function

Fetches the specified number of characters from a specified location of the exclusive character variable (\$).

Parameter

Location: Numerical expression. Values below the decimal point are discarded.

$$1 \leq \text{location} < 101$$

Number of characters: Numerical expression. Values below the decimal point are discarded.

$$1 \leq \text{number of characters} < 101$$

When omitted, all characters after the specified location are fetched.

Explanation

- (1) Fetches a specified number of characters from a specified location of the exclusive character variable (\$).
- (2) When the specified location is out of the character string, a null is obtained.
- (3) When the length of the character string after the specified location is smaller than the specified number of characters, all the characters after the specified location are fetched.

* MID\$ can be abbreviated as MID.

Example

```
10 $="ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
20 INPUT M,N  
30 PRINT MID$(M,N)  
40 END
```

VAL

(Simple character variable)

Ⓕ

Function

Converts characters in a simple character variable into a numerical value.

Parameter

Simple character variable: An array variable cannot be used.

Explanation

- (1) Converts characters in a simple character variable into a numerical value.
- (2) When the content of a character variable includes +, -, *, E or E-, it is converted into a numerical value as it is.

When A\$ = "-12.3 ", VAL(A\$) → -12.3

- (3) When the content of a character variable starts with a character other than a numeral, +, -, or *, an error occurs.

When A\$ = " A45 ", VAL(A\$) → - error (ERR2)

- (4) When a character other than a numeral is inserted in the middle, only the part before this character is converted to a numerical value.

When A\$ = " 78A9 ", VAL(A\$) → 78

Example

```
10 INPUT A$  
20 PRINT VAL(A$)  
30 END
```

STR\$ (Numerical expression)

(F)

Function

Converts the value of a numerical expression into a character string.

Parameter

Numerical expression: Numerical value, calculation expression, numerical variable, numerical array variable.

Explanation

- (1) Converts the value of a numerical expression into a character string.
- (2) When the numerical expression is a calculation expression, the calculation result is converted into a character string.
- (3) When a numerical expression is positive, the sign digit is deleted and only the numerals are converted.

Example

```
10 PRINT STR$(123)
20 PRINT STR$(45+78)
30 A=963
40 PRINT STR$(A)
50 END
```

NUMERICAL FUNCTIONS

SIN

Argument
Numerical expression

COS

Argument
Numerical expression

(F)

TAN

Argument
Numerical expression

Function

Obtains the value of a trigonometric function for a given argument.

Parameter

Argument: Numerical expression

$-1440^\circ < \text{argument} < 1440^\circ$ (degrees)

$-8\pi < \text{argument} < 8\pi$ (radians)

$-1600 < \text{argument} < 1600$ (grades)

However, for TAN, " $|\text{Argument}| = (2n-1) * 1 \text{ right angle}$ " is excluded.

$1 \text{ right angle} = 90^\circ = \frac{\pi}{2} \text{ rad} = 100 \text{ grad}.$

Explanation

- (1) Obtains the value of a trigonometric function for a given argument.
- (2) The value depends on the angle unit setting (by the **MODE** key or MODE command).

ASNArgument
Numerical expression**ACS**Argument
Numerical expression

Ⓕ

ATNArgument
Numerical expression**Function**

Inverse trigonometric function that obtains an angle for a given argument.

Parameter

Argument: Numerical expression.

For ASN, ACS, $-1 \leq \text{argument} \leq 1$.

Explanation

- (1) Inverse trigonometric function that obtains an angle for a given argument.
- (2) The value depends on the angle unit setting (by the **MODE** key or MODE command).
- (3) The values of the functions are given within the following range.

$$-90^\circ \leq \text{ASN } X \leq 90^\circ$$

$$0^\circ \leq \text{ACS } X \leq 180^\circ$$

$$-90^\circ \leq \text{ATN } X \leq 90^\circ$$

LOGArgument
Numerical expression**LN**Argument
Numerical expression

Ⓕ

Function

Gives the value of a logarithmic function.

Parameter

Argument: Numerical expression.

$$0 < \text{argument}$$

Explanation

Gives the value of a logarithmic function.

- LOG Common logarithmic function $\log_{10}x, \log x$
- LN Natural logarithmic function $\log_e x, \ln x$

EXPArgument
Numerical expression

Ⓕ

Function

Gives the value of an exponential function.

Parameter

Argument: Numerical expression.

$$-227 \leq \text{argument} \leq 230$$

Explanation

Gives the value of an exponential function.

$$\text{EXP } e^x$$

SQRArgument
Numerical expression

Ⓕ

Function

Gives the square root of an argument.

Parameter

Argument: Numerical expression.

$$0 \leq \text{argument}$$

Explanation

Gives the square root of an argument.

$$\text{SQR } \sqrt{x}$$

ABS

Argument
Numerical expression

Ⓕ

Function

Gives the absolute value of an argument.

Parameter

Argument: Numerical expression.

Explanation

Gives the absolute value of an argument.

ABS $|x|$

SGN

Argument
Numerical expression

Ⓕ

Function

Gives a value that corresponds to the sign of an argument.

Parameter

Argument: Numerical expression.

Explanation

Gives a value that corresponds to the sign of an argument.

When an argument is positive, 1

When an argument is 0, 0

When an argument is negative, -1

INT

Argument
Numerical expression

Ⓕ

Function

Gives the maximum integer that does not exceed an argument.

Parameter

Argument: Numerical expression.

Explanation

Gives the maximum integer that does not exceed an argument.

INT 12.56 $\rightarrow$ 12

INT -78.1 $\rightarrow$ -79

FRAC

Argument
Numerical expression

Ⓕ

Function

Gives the decimal part of an argument.

Parameter

Argument: Numerical expression.

Gives the decimal part of an argument. The sign is in accordance with the sign of the argument.

RND

(Argument
Numerical expression , digit location)
Numerical expression

Ⓕ

Function

Gives the value of an argument which is rounded off at the specified location.

Parameter

Argument: Numerical expression.

Location: Numerical expression. Values below the decimal point are discarded.

-100 < location < 100

Explanation

(1) Gives the value of an argument which is rounded off at the specified location.

(2) The argument is rounded off at the 3rd decimal place (10^{-3}).

→ **RND** (x, -3)

The argument is rounded off at the place of 100s (10^2).

→ **RND** (x, 2)

RAN

Ⓕ

Function

Gives a random number from 0 to 1.

Explanation

(1) Gives a random number from 0 to 1.

$$0 < \text{random number} < 1$$

(2) The random number has 10 digits.

Example

Provides a random number with 1 digit from 0—9.

$$\text{INT}(\text{RAN} \# * 10)$$

Provides a random number with 1 digit from 1—5.

$$\text{INT}(\text{RAN} \# * 5) + 1$$

Provides a random number with 2 digits from 10—99.

$$\text{INT}(\text{RAN} \# * 90) + 10$$

DEG

(Degree [, Minute [, Second]])
Numerical expression Numerical expression Numerical expression

Ⓕ

Function

Converts sexagesimal to decimal.

Parameter

Degree: Numerical expression.

Minute: Numerical expression.

Second: Numerical expression.

$$|\text{DEG}(\text{degree, minute, second})| < 10^{100}$$

Explanation

Converts sexagesimal expressed by degree, minute, and second to decimal.

Example

DEG(12, 34, 56) **EXE**

12.58222222

10 INPUT A,B,C

20 PRINT DEG(A,B,C)

30 END

DMSS

Argument
Numerical expression

Ⓕ

Function

Converts decimal to sexagesimal.

Parameter

Argument: Numerical expression.

$$|\text{numerical expression}| < 10^{100}$$

Explanation

(1) Converts decimal to sexagesimal.

(2) The converted result is provided as a character string.

Example

DMSS(45.678) **EXE**

45°40'40.8

10 INPUT A

20 \$=DMSS(A)

30 PRINT\$

40 END

DATA BANK COMMANDS

NEW

(M)

Function

Erases data for Data Bank function.

Explanation

- (1) Erases all stored data.
- (2) Cannot be executed when a password is specified.
- (3) Can only be executed in the WRT mode.

Example

MODE 1

NEW # EXE

MODE 2

LIST

(M)

Function

Displays all data for Data Bank function.

Explanation

- (1) Displays stored data sequentially from the beginning.
- (2) Displayed contents are a sequential No. and data.
- (3) Since data are automatically displayed sequentially, press the **STOP** key to stop display. Press the **EXE** key to resume the display of the next data.
- (4) In the Print mode (MODE 2), the display is not stopped but is performed sequentially at high speed.
- (5) Cannot be executed when a password is specified.
- (6) Cannot be executed in the input mode for Data Bank function (MODE 3).

Example

LIST # EXE

SAVE

["File name"]
Character string

(M)

Function

Stores data for Data Bank function on a cassette tape.

Parameter

File name: A string with 1—8 characters. Can be omitted.

Explanation

- (1) Stores data on a cassette tape.
- (2) Since data for Data Bank functions cannot be stored with SAVE or SAVE ALL, be sure to use SAVE #.
- (3) If a password has been specified, storing is performed with this password. Therefore, the same password must be specified when the loading is performed by the LOAD # command.
- (4) Cannot be executed in the input mode for Data Bank function.

Example

SAVE # EXE

SAVE # "CASIO" EXE

LOAD

["File name"]
Character string

(M)

Function

Loads data for Data Bank function from a cassette tape.

Parameter

File name: A string with 1—8 characters. Can be omitted.

Explanation

- (1) Loads data stored on a cassette tape.
- (2) When data stored with a password are loaded, this stored password must be specified.
- (3) If data exist in the computer, new data are loaded after existing data are cleared.
- (4) Cannot be executed in the input mode for Data Bank function.

Example

LOAD # EXE

LOAD # "CASIO" EXE

READ# Variable name [, variable name]*

(P)

Function

Reads data for Data Bank function.

Parameter

Variable name: Numerical variable or character variable.
An array variable can also be used.

Explanation

- (1) Sequentially reads stored data to a variable.
- (2) Only numerical type data can be read for a numerical variable. If character type data are used, an error (ERR2) occurs.
- (3) After the necessary data are read by a READ# statement, the following data are read by the next READ# statement.
- (4) When data are punctuated by " , " , they are read in the order in which they are written.

Example) DATA

No. 1 A, X, Y
No. 2 B, Z
No. 3 C



Reading sequence

A→X→Y→B→Z→C

- (5) When data to be read does not exist, an error (ERR4) occurs.
- (6) The data sequence to be read can be modified by RESTORE# (see page 183).
- (7) When a space exists at the beginning of a data, it is skipped.
- (8) When data is inside " ", the character string inside " " is read.

Example

<Data>

No. 1 1, 2, 3
No. 2 4, 5, 6
No. 3 7, 8, 9
No. 4 10,

<Program>

```
10 A=0
20 READ# $
30 IF $=" " THEN 60
40 A=A+VAL($ )
50 GOTO 20
60 PRINT "Σx=";A
70 END
```

- Reads numerical data to obtain a sum.

RESTORE# ["Searched character string" [Character expression] [{ 0 }] [{ Line number } # program area number]]

(P)

Function

Searches data for Data Bank function and specifies the sequence of the data to be read by READ#.

Parameter

Searched character string: Character expression. When a character string is used, place it inside " " .
Line number: Numerical expression.

$0 < \text{line number} < 10000$

Program area No.: Numerical expression.

$0 \leq \text{program area No.} < 10$

Explanation

- (1) Searches data and specifies the sequence of data to be read by the following READ# statement.
- (2) The relationship between a parameter and data searching is as follows.

① RESTORE

When the searched character string and after are omitted, data are read from the beginning by the following READ#.

② RESTORE # "searched character string"

Searches data that begins with the searched character string, and this data is read by the following READ#.

③ RESTORE # "searched character string", { 0 }

When 0 is specified, it is the same as ②.

When 1 is specified, the first data of the line that includes searched data is read by the following READ# statement.

④ RESTORE # "searched character string", [{ 0 }] ,

{ line number }
program area No. }

When executing searching, it branches to the specified line or a program area if appropriate data does not exist.

- * In ② and ③, when appropriate data does not exist, an error (ERR4) occurs.
- * In ④, when a branching line number does not exist, or when a program does not exist in the program area, an error (ERR4) occurs.

Example

<Data>

No.1 FOSTER,347-4811,NEW YORK
No.2 SMITH,045-211-0821,CHICAGO
No.3 JONES,06-314-2681,SAN FRANCISCO
No.4 BROWN,075-351-1161,LOS ANGELES

<Program>

```
10 RESTORE#  
20 READ# $  
30 PRINT $  
40 RESTORE#"S"  
50 READ# $  
60 PRINT $  
70 RESTORE#"LO",1  
80 READ# $  
90 PRINT $  
100 RESTORE#"AA",1,200  
110 READ# $  
120 PRINT $  
130 END  
200 PRINT"END"  
210 END
```

} Data stored at the beginning is displayed.

} Data whose initial letter is S is displayed

} Searches data whose initial two letters are LO, and displays the first data on the line which includes the data.

} When data whose initial two letters are AA does not exist, branching to line 200 is executed.

RUN EXE

EXE
EXE
EXE

FOSTER
SMITH
BROWN
END

WRITE

[Data [, Data]*]
expression expression

(P)

Function

Rewrites or deletes data for Data Bank function.

Parameter

Data: Numerical expression or character expression. When a character string is used, place it inside " ".

Explanation

- (1) Writes data in the record area currently specified by RESTORE#.
- (2) Data are newly written without any relationship to data existence in the appropriate record area.
- (3) When no data is specified, stored data in the record area are deleted.
- (4) When plural data exist, these data can be written on the same record area by using " , " for punctuation.
- (5) After the necessary data are written by the first WRITE # statement, the following data are written by the next WRITE # statement.

Example

```
10 REM WRITE  
20 RESTORE#  
30 WRITE#"A,B,C"  
40 RESTORE#  
50 FOR I=1 TO 3  
60 READ# $:PRINT $  
70 NEXT I  
80 PRINT" "  
90 REM CHANGE  
100 RESTORE#  
110 FOR I=1 TO 3  
120 WRITE# STR$(I)  
130 NEXT I  
140 RESTORE#  
150 FOR I=1 TO 3
```

} New data is written.

} Data is rewritten.

```

160 READ# $: PRINT $;
170 NEXT I
180 PRINT " "
190 REM CLEAR
200 RESTORE#
210 WRITE#
220 RESTORE#
230 READ# $

```

— Data erase

Operation

RUN **EXE**
EXE
EXE

Display

ABC
123
ERR4 P0-230

↓
Shortage of data due to data erase.

CHAPTER 6

REFERENCE MATERIAL

6-1. ERROR MESSAGE TABLE

Error code	Meaning	Cause	Corrective measure
1	Memory overflow or operator level overflow.	- Number of steps are insufficient. Program cannot be written. - Operator level overflow	- Clear unnecessary programs or reduce the number of memories. - Divide the formulas and make them simpler.
2	Syntax error	- Format error in program, etc. - Left-hand and right-hand formats differ in an assignment statement, etc.	Correct error in input program, etc.
3	Mathematical error	- The result of a numerical expression calculation exceeds $\pm 1 \times 10^{100}$. - The argument of numerical function is outside the input range. - Result is indefinite or impossible.	- Correct the calculation formula or data. - Verify the data.
4	Undefined error	- No designated line number for GOTO statement or GOSUB statement. - A READ or READ# statement is executed when there is no data to be read.	- Designate the line number. - Check the relation between the READ and DATA statements. Create data in the statement to be assigned to the variable.
5	Argument error	For a command or function that requires an argument, the argument is outside the input range.	Correct the argument.
6	Variable error	- Attempt was made to use a memory which has not been expanded. - Attempt was made to use the same memory for a numerical variable and a character variable at the same time.	- Expand the memory properly. - Do not use the same memory for a numerical variable and a character variable at the same time.
7	Nesting error	- RETURN statement is executed when subroutine is not being executed. - NEXT statement is executed when not in FOR loop. - Subroutine nesting levels exceed 8. - FOR-NEXT loop nesting levels exceed 4.	- Remove unneeded RETURN statements or NEXT statements. - Keep the subroutines or FOR-NEXT statement loops within required levels.

6-1 ERROR MESSAGE TABLE

Error code	Meaning	Cause	Corrective measure
8	Password	When the password is specified, a) another password is specified. b) a command such as LIST or NEW which can not be used is executed.	Cancel the password error by entering the correct password.
9	Option error	- SAVE or PUT command was executed without a tape recorder being connected. - Input signal used by LOAD or GET command cannot be read. - Printer battery is weak. - Printer paper jammed.	- Connect a tape recorder. - Lower the tape recorder volume. - Set the tone control of the tape recorder to middle position. - Replace the tape. - Clean the tape recorder heads - Charge the battery. - Unjam the printer paper.

6-2. CHARACTER CODE TABLE

	SPACE	+	-	*	/	↑	↓	"	#	\$	>	≥	=	≤	<	≠
Numbers	0	1	2	3	4	5	6	7	8	9	.	π	)	(	E	E
Capital letters	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Small letters	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
Symbols	?	,	:	:												
Graphic symbols	○	Σ	°	△	@	×	÷	♠	←	♥	♦	♣	μ	Ω	↓	→
	%	¥	□	[	&	-	'	.	]	■	\					

The characters and symbols in the above table are lined in sequence, with SPACE being the smallest and " \ " being the largest. (" \ " can be displayed by pressing **SHIFT** **P**.)

COMMAND/FUNCTION INDEX

ABS	32, 176	MID\$	85, 170
ACS	29, 174	MODE	167
ASN	29, 174	NEW(ALL)	45, 143
ATN	29, 174	NEW #	56, 180
BEEP	81, 165	ON-GOSUB	84, 160
CLEAR	149	ON-GOTO	84, 156
COS	30, 173	PASS	146
CSR	87, 154	PRINT	36, 60, 153
DATA	82, 161	PUT	94, 164
DEFM	80, 166	RAN #	32, 178
DEG	33, 178	READ	82, 162
DMS\$	34, 179	READ #	182
END	42, 150	REM	151
EXP	31, 175	RESTORE	82, 163
FOR-TO-STEP/NEXT	68, 158	RESTORE #	183
FRAC	32, 177	RETURN	71, 160
GET	94, 164	RND	32, 177
GOSUB	71, 159	RUN	49, 144
GOTO	58, 155	SAVE(ALL)	91, 147
IF-THEN	62, 157	SAVE #	93, 181
INPUT	36, 151	SET	33, 168
INT	32, 176	SGN	32, 176
KEY\$	87, 152	SIN	30, 173
LEN	85, 169	SQR	31, 175
LET	38, 150	STOP	55, 150
LIST	96, 145	STR\$	85, 172
LIST #	180	TAN	30, 173
LN	31, 174	VAL	85, 171
LOAD(ALL)	91, 148	VERIFY	149
LOAD #	93, 181	WRITE #	185
LOG	31, 174		

SPECIFICATIONS

■ Type

PB-120

■ Fundamental calculation functions

Negative numbers, exponents, parenthetical addition, subtraction, multiplication and division (with priority sequence judgement function (true algebraic logic))

■ Built-in functions

Trigonometric/inverse trigonometric functions (angular units — degree/radian/grade), logarithmic/exponential functions, square roots, powers, conversion to integer, deletion of integer portion, absolute value, symbolization, designation of number of significant digits, designation of number of decimal digits, random numbers, π , decimal $\leftrightarrow$ sexagesimal conversion.

■ Commands

INPUT, PRINT, GOTO, ON-GOTO, FOR-NEXT, IF-THEN, GOSUB, ON-GOSUB, RETURN, READ, DATA, RESTORE, STOP, END, REM, LET, BEEP, PASS, RUN, LIST, LIST ALL, MODE, SET, CLEAR, NEW, NEW ALL, DEFM, SAVE, SAVE ALL, LOAD, LOAD ALL, PUT, GET, VERIFY, NEW #, LIST #, LOAD #, SAVE #, READ #, WRITE #, RESTORE #.

■ Program functions

KEY\$, CSR, LEN, MID\$, VAL, STR\$

■ Calculation range

$\pm 1 \times 10^{-99}$ to $\pm 9.999999999 \times 10^{99}$ and 0 (internal calculations use 12-digit mantissa)

■ Program system

Stored system

■ Number of steps

Maximum 7648 steps

■ Program capacity

Maximum 10 programs (P0 through P9)

■ Number of variables

Standard 26, expandable to 982 and exclusive character variable (\$)

■ Nesting

Subroutine — 8 levels

FOR-NEXT loop — 4 levels

Numerical value — 6 levels

Operators — 12 levels

■ Display system and contents

10-digit mantissa (including minus sign) or 8-digit mantissa (7 digits for negative number) and 2-digit exponent.

SPECIFICATIONS

- **Display elements**

- 12-digit dot matrix display (liquid crystal)

- **Main components**

- C-MOS VLSI and others

- **Power supply**

- Main power supply — Two CR2032 lithium batteries

- Memory backup power supply — One CR1220 lithium battery

- **Power consumption**

- 0.07W

- **Battery life**

- Main power supply

- *Approximately 130 hours (continuous program execution).

- *Approximately 65 hours (with options (FA-12/FA-3) connected.)

- Memory backup power supply

- *Approximately two years.

- **Auto power-off**

- Power is turned off automatically approximately 6 minutes after last operation.

- **Ambient temperature range**

- 0°C to 40°C (32°F to 104°F)

- **Dimensions**

- 9.8H × 165W × 71mmD ($\frac{3}{8}$ "H × 6 $\frac{1}{2}$ "W × 2 $\frac{3}{4}$ "D)

- **Weight**

- 124g (4.4 oz) including batteries